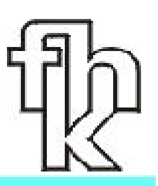
**Fachhochschule Konstanz**
Hochschule für Technik, Wirtschaft und Gestaltung
**University of Applied Sciences**



# ED_TRA

# Engineering Desktop Anwendungen in der Tragwerksplanung

**Engineering Desktop Applications in Structural Design**

**gefördert vom BMBF im Programm aFuE**

# Schlussbericht

**Fachhochschule Konstanz**

**Januar 2002**

# ED_TRA

# Engineering Desktop Anwendungen in der Tragwerksplanung

**Engineering Desktop Applications in Structural Design**

**Prof. Dr.-Ing. Horst Werkle**

**Prof. Dr. H. Pleßke**

**Mitarbeit**

**Andreas Bitzer**
**Indira Maharashvili**
**Joachim Ritter**
**Heinrich Rotgang**
**Dorothee Wientges**

**Förderung: BMBF FKZ 17.064.99**

# Inhalt

## Teil I

# Teil II

**II-2 Softwarekomponenten**

## II-4 Handbücher für die Softwarebenutzung

## I-1 Aufgabenstellung und Ziele des Projekts

Statische Berechnungen werden heute vorwiegend unter Verwendung von Standardsoftware erstellt. Im wesentlichen kommen dabei Programmsysteme zum Einsatz, die für spezielle Aufgaben konzipiert sind. Beispielweise gibt es zur Berechnung von Decken, Stützen und Dachkonstruktionen im Hochbau entsprechende Stabwerks- und Finite-Element-Programme. Bei der Tragwerksplanung sind jedoch auch Berechnungen durchzuführen, für die fertige Programme nicht zur Verfügung stehen. Diese werden meistens“ von Hand“ , d.h. mit Papier, Bleistift und Taschenrechner durchgeführt. Hierbei kann es sich um spezielle statische Nachweise, z. B. für besondere Bauteile wie Dübelverankerungen, Fundamentanschlüsse für Stützen, ausgeklinkte Träger u.ä. handeln. Aber auch einfache Kontrollen von Computerberechnungen und Vordimensionierungen komplizierter Systeme erfolgen heute noch in der Regel „von Hand“. Zur Durchführung solcher Berechnungen auf dem Computer fehlen heute zwar nicht mehr die allgemeinen Werkzeuge, wohl aber geeignete Software-Hilfsmittel, wie sie bei der Handrechnung etwa die einschlägigen Tabellenbücher darstellen. Diese sind aber wesentlich für eine Engineering-Desktop Anwendung auf dem Computer. Diese zeichnet sich durch eine durchgängige Verwendung des Rechners auch bei nicht standardisierten Berechnungsaufgaben im Bauingenieurwesen aus.

Ziel des Projekts war die Entwicklung von Softwarebausteinen für integrierte Engineering-Desktop-Anwendungen für die Tragwerksplanung. Die Funktionalität des Softwarebaukastens lässt sich aus dem statischen ”Wissen” eines Handbuchs wie z.B. [1] ableiten. Die Softwarebausteine sind so aufgebaut, dass sie von verschiedenster Officesoftware, wie MS-Excel und Mathcad genutzt werden können. Mit diesen Bausteinen soll eine deutlich höhere Flexibilität bei der Führung statischer Nachweise erreicht werden als dies bei der derzeitigen Standardsoftware für vorgegebene Nachweisabläufe der Fall ist. Mit Hilfe der entwickelten Softwarebausteine wurde eine Implementierung in Mathcad und Excel vorgenommen.

## I-2 Voraussetzungen für das Vorhaben

Ausgangsbasis des Vorhabens war eine Vorstudie mit Mathcad. Hierbei wurden in C Funktionen entwickelt, die es ermöglichen in einfachen Fällen mit Mathcad Tragwerksplanung zu betreiben. Diese Arbeiten wurden im vorliegenden Forschungsprojekt auf eine breitere und von Mathcad unabhängige Basis gestellt. Weiterhin wurden im Rahmen des Projekts eine Diplomarbeit im Fachbereich Bauingenieurwesen und zwei studentische Projekte im Fachbereich Informatik durchgeführt.

Voraussetzung des Vorhabens war die Genehmigung des vorliegenden Forschungsvorhabens. Weitere materielle Unterstützung erfuhr das Projekt durch das Institut für Angewandte Forschung der Fachhochschule Konstanz. Zwei Lizenzen von Mathcad2000 wurden von der Firma Mathsoft, München, unentgeltlich zu Verfügung gestellt.

## I-3 Planung und Ablauf des Projekts

Die Durchführung des Projekts war in folgenden Schritten geplant:

1. Entscheidung für die Softwaretechnologie und Konzeptentwicklung
2. Entwicklung eines Katalogs von Funktionen für die Tragwerksplanung
3. Umsetzung der Funktionen im JAVA-Server
4. Oberflächenentwicklung für Excel in VBA
5. Anbindung an Mathcad mit C++
6. Tests und Beispielrechnungen

Der Ablauf folgte den geplanten Schritten. Allerdings wurden die Schritte 3 bis 5 teilweise parallel realisiert. Die Gründe hierfür lagen einerseits in der Notwendigkeit, die Durchgängigkeit des Softwarekonzepts sicherzustellen sowie andererseits in einem sinnvollen Personaleinsatz bei mehreren Mitarbeitern.

# I-4 Wissenschaftlicher und technischer Stand der Entwicklung

## 4.1 Allgemeiner Stand der Entwicklung

Ziel ist die durchgängige Verwendung des Computers auch bei nicht standardisierbaren Berechnungsaufgaben. Bei allgemeiner Bürosoftware im Office-Bereich ist dies mit Textverarbeitung, Tabellenkalkulation und Datenbanken bereits seit längerem Stand der Technik. Im Ingenieurbereich gibt es hierzu folgende Ansätze:

- Baustatik-Editoren: Hierbei handelt es sich um eigenständige Softwareprodukte, die speziell für die Erstellung von statischen Berechnungen entwickelt wurden.
- Anwendungen für MS-Excel: Dies sind Arbeitsblätter und Arbeitsmappen, die als Vorlage für statische Berechnung und Nachweise genutzt werden können.
- Hilfsmittel für Mathcad: Statische Berechnungen mit Mathcad werden durch Elektronische Bücher, Arbeitsblätter für spezielle Aufgabenstellungen und kleine Programme unterstützt.
- Neuere Ansätze mit MS-Word, die auf der Komponententechnologie beruhen.

Im folgenden werden die einzelnen Ansätze diskutiert.

## 4.2 Baustatik-Editoren

Baustatik-Editoren wurden entwickelt um die für spezielle Aufgaben verfügbaren Statik-Programme unter einer einheitlichen Benutzeroberfläche zu integrieren und um einen statischen Bericht zu verfassen. Einen Überblick gibt Anhang A.

Manche Statikeditoren sind spezielle Texteditoren zur Erstellung statischer Berechnungen. Sie enthalten auch Rechenteile, die es ermöglichen, Formeln einzugeben und einfache Berechnungen durchzuführen. Für Standardaufgaben liegen auch vorbereitete Arbeitsblätter vor.

Bei den integrierten Statikeditoren handelt es sich um eine Benutzeroberfläche zum Aufruf von Statikprogrammen für Standardaufgaben. Sie erlauben es, bereits gerechnete Statikpositionen durch Einfügen zusätzlicher Kommentare und Hinweise für den Ausdruck aufzubereiten.

Baustatik-Editoren verfügen nicht über OLE-Fähigkeiten und nur sehr eingeschränkte allgemeine Rechenfunktionalität. Sie haben zwar den Anspruch, eine Desktop-Oberfläche für Ingenieure zu bieten, können diesen Anspruch aber kaum erfüllen, da sie naturgemäß als eigenständige Programmentwicklungen nur eine bedingte Flexibilität und Funktionalität aufweisen.

### 4.3 Tabellenkalkulationsprogramme

Für Tabellenkalkulationsprogramme wie Microsoft Excel existieren eine Vielzahl Arbeitsblätter für die Tragwerksplanung. Die Berechnung im einfachen Fall innerhalb der Zellen der Arbeitsblätter erfolgen oder über ein Steuerelement[1] wird eine Berechnungsroutine gestartet, die innerhalb eines Makros[2] ausgeführt wird. Einen Überblick gibt Anhang B.

Tabellenkalkulationsprogramme sind für statische Berechnungen grundsätzlich geeignet und werden in der Tragwerksplanung auch für tabellarische Zusammenstellungen großer Datenmengen eingesetzt. Sie sind in der Regel OLE-fähig und ermöglichen eine große Flexibilität. Nachteilig ist, dass die zugrunde liegenden Formeln nicht im Ausdruck sichtbar sind. Weiterhin fehlen Funktionen, die für statische Berechnungen auf einfache Weise aufgerufen werden können.

### 4.4 Mathcad

Mathcad ist ein Computeralgebrasystem, das für ingenieurmäßige Berechnungen entwickelt wurde. Die verwendeten Variablen und Formeln werden in mathematischer Notation im Arbeitsblatt angezeigt. Zur Unterstützung bei Aufgaben der Tragwerksplanung stehen Arbeitsblätter und sogenannte „Elektronische Bücher“ zur Verfügung, in denen entsprechende Formeln bereits enthalten sind.

Als vorteilhaft erweist sich bei Mathcad die hohe Flexibilität bei der Berechnung, die umfangreiche mathematische Funktionalität, die selbsterklärende Notation und die Integrationsfähigkeit über OLE-Techniken. Für ingenieurmäßige Berechnungen im Rahmen einer Engineering-Desktop-Umgebung ist Mathcad daher bestens geeignet.

In neueren Arbeiten wurden auch spezielle Funktionen für die Tragwerksplanung in C++ entwickelt. Dieser Ansatz wird beim vorliegenden Forschungsvorhaben in einer allgemeineren, von Mathcad unabhängigen Weise, fortgeführt.

### 4.5 Word mit Komponententechnologie

Die Komponententechnologie ermöglicht es, bei der Softwareentwicklung Klassenbibliotheken zu nutzen, die in einer weitgehend beliebigen Programmiersprache entwickelt wurden. In [9] wird ein Konzept vorgestellt und in einzelnen Beispielen implementiert, bei dem Statik-Komponenten in Visual Basic entwickelt wurden und von WORD mit „Visual Basic for Applications“ genutzt werden. Dieses Konzept ist, da WORD über keine eigene Rechenfunktionalität verfügt, auf Standardaufgaben der Tragwerksplanung beschränkt. Sehr gut ist die Integration in ein bestehendes Textverarbeitungssystem als Container gelöst.

[1] Steuerelemente wie z.B. Befehlsschaltflächen, beispielsweise aus der Microsoft Forms Bibliothek

[2] Makro ist die Bezeichnung für Anweisungen, seit MS-Excel 5.0 in der Sprache Visual Basic for Applications, die in Modulen gespeichert werden können und nach dem Aufruf während der Laufzeit interpretiert werden

Die Komponententechnologie wird auch im vorliegenden Forschungsvorhaben genutzt, allerdings mit einem allgemeineren, von Visual Basic unabhängigen Ansatz.

## Literatur

[1] Schneider K.J.
Bautabellen für Ingenieure
Werner-Verlag, Düsseldorf, 13. Auflage 1998

[2] Schulz G.
Software zum Test neuer Berechnungsalgorithmen
Bauinformatik-Journal 1, 1998, Günther Schulz Verlag, Celle, 1998

[3] Werkle, H.
Die FEM in der anwendungsorientierten Lehre an Fachhochschulen
3.FEM/CAD-Tagung, TH Darmstadt, Darmstadt 1994

[4] Werkle H.
Handbuch unifem, unicad, Hochtief AG. Frankfurt, 1991

[5] Werkle H.
Der CAD-Plan als digitales Gebäudemodell für baustatische Berechnungen
Abschlußbericht, FH Konstanz, Konstanz 1993

[6] Werkle, H., Hansen, R., Röder, J.
Object-Oriented Databases in Software Development for Structural Analysis.
International Converence on the Applications of Computer Science and Mathematics in Architecture and Civil Engineering.
Weimar: Bauhaus-Universität, Weimar 1997

[7] Pleßke H., Die Anwendung von Mathcad zur Darstellung wirtschaftsmathematischer Fragestellungen.
LARS-Workshop: Computeralgebra in Lehre und Forschung
FH Heilbronn, Künzelsau 1995

[8] Pleßke H.
Die Visualisierung mathematischer Gegenstände mittels Computeranimation
LARS-Workshop: Innovative Lehr- und Lernmittel
FH Konstanz 1998

[9] Bittrich D.
Verbunddokumente als Nutzeroberfläche von Software für die Tragwerksplanung, IKM 2000, Bauhaus-Universität Weimar, Juni 2000

# II-1 Softwarekonzept

Das Kapitel gibt einen ersten Überblick über die erreichten direkten Ziele, die im vorangehenden Kapitel erläutert wurden und stellt ebenfalls einen Einblick in die Programmiersprachen dar, die Verwendung fanden oder für deren Verwendung es Überlegungen gab.

## 1.1 Gesamtkonzept

Aus den im vorigen Kapitel genannten Zielen lässt sich nach einer Evaluierung ein Gesamt- oder Grundkonzept ableiten, das im nachfolgenden Schaubild dargestellt ist und in den folgenden Kapiteln kurz erläutert wird.

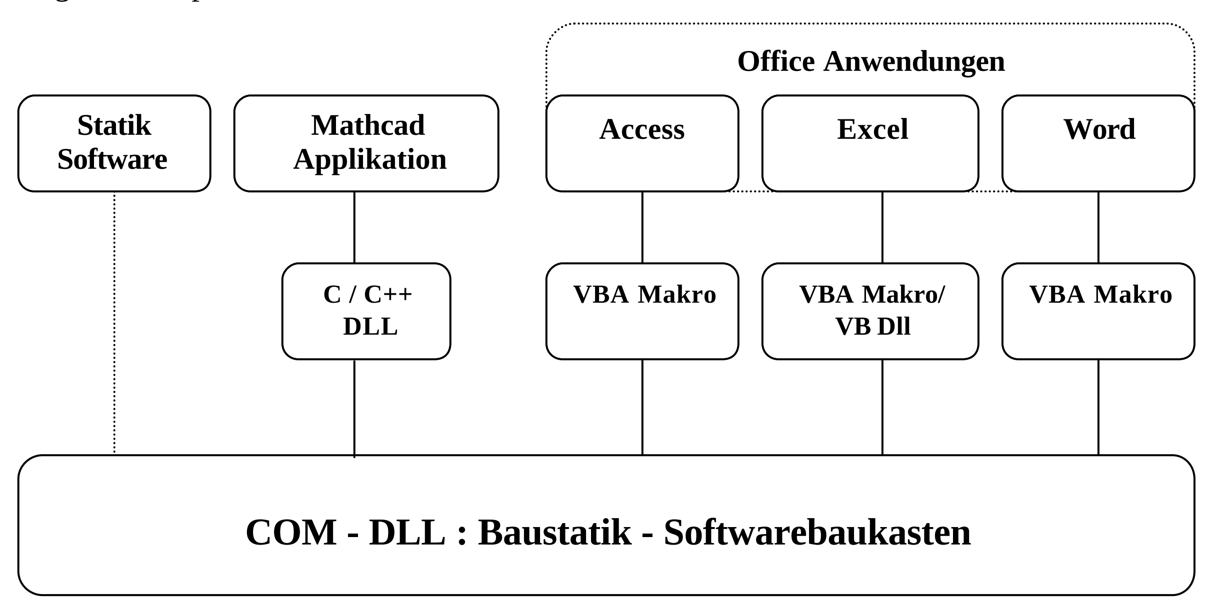



Bild 1. Schema des Aufbaus des Projekts

Da im Bereich der Softwaretechnologie derzeit kein einheitlicher Standart besteht, was das zugrundeliegende Betriebssystem betrifft und hier bis jetzt jede Firma ihre eigene Technologie entwickelt und eigene Konzepte verfolgt, galt es im Vorfeld zu Überlegen, für welche Betriebssysteme die Umsetzung eines Engineering-Desktop überhaupt sinnvoll erscheint oder notwendig ist. Dennoch wollte man bei der Erstellung der Funktionen des Softwarebaukastens so allgemein wie möglich bleiben, um auch für zukünftige Entwicklungen in diesem Bereich offen zu sein.

Im Bereich der Erstellung von statischen Berechnungen im Bauwesen ist derzeit das Betriebssystem Windows der Firma Microsoft am stärksten verbreitet. Daher wurde eine Auslegung des Projekt auf dieses Betriebssystem gewählt.

Um für zukünftige Entwicklungen offen zu bleiben, sollten die Funktionen des Softwarebaukastens in einer Programmiersprache implementiert werden, welche es erlaubt die Funktionen eventuell auch auf andere Betriebssysteme zu portieren, also den Quellcode portabel machen. Idealerweise bietet sich hierfür die Programmiersprache Java[1] an.

[1] Siehe hierzu auch Kapitel 1.3

Da das gewählte Betriebsystem, in Bezug auf die Technologie, welche die Anbindung der Komponenten oder auch In-Process-Server[2], an andere Programme überhaupt ermöglicht, bestimmte Voraussetzungen erfordert, was im nächsten Kapitel näher erläutert wird, konnte die Absicht die Funktionalität auf anderen Betriebssystemen zu ermöglichen, softwaretechnologisch bedingt, nicht durchgeführt werden. Durch die Entscheidung, die Komponenten in der Sprache Java zu entwickeln, blieb zumindest der Quellcode im wesentlichen portabel.

## 1.2 COM[3] Technologie

Die Technologie, welche sich hinter der Möglichkeit der Nutzung von Komponenten als Server verbirgt, nennt sich COM, sofern man sich auf dem Betriebssystem Windows/NT der Firma Microsoft bewegt. Bevor näher auf die programmierten Komponenten eingegangen wird, ist es sinnvoll, vorab den Begriff COM und diese zugrundeliegende Technologie näher zu erläutern.

Das Component Object Mode, kurz COM, legt fest, auf welche Weise binäre Softwaremodule zusammenarbeiten. Es besteht aus einer Reihe von Systemdateien, die auf einer unteren Ebene des Betriebssystems für die Zusammenarbeit sorgen. COM ist somit allzeit in Windows präsent und stellt nach Meinung mancher Fachleute die Zukunft der Systemprogrammierung unter Windows dar [1].

Heute besteht die große Mehrheit des Betriebssystems Windows jedoch noch auf Systemdateien, die ihre Funktionalität über API[4]-Aufrufe zur Verfügung stellen. Wird der Code zu einer DLL in einer geeigneten Programmiersprache geschrieben und kompiliert, wird ein Programm erstellt, das anderen Programmen eine oder mehrere COM-Komponenten oder COM-Objekte zur Verfügung stellt.

Auch alle Systemfunktionen die Programmierern zur Verfügung stehen sind in verschiedenen Programmbibliotheken (*DLL's*) gespeichert. Die Gesamtheit dieser Bibliotheken wird Windows-API genannt. Eine Liste der wichtigsten Programmbibliotheken ist im Anhang zu finden. Mit der Verwendung von API-Funktionen bei der Programmierung bewegt man sich auf einer systemnahen Ebene. Man greift über API Funktionen beispielsweise auf Betriebssystemfunktionen oder auch auf eigene programmierte Funktionen zu, was letztlich auch wieder nur das Aufrufen von DLL's bedeutet. Ein wesentlicher Vorteil ist die Geschwindigkeit mit der auf dieser Ebene Daten verarbeitet werden können, da man sich auf einer systemnahen Ebene bewegt.

Komponenten, die als Server dienen, was auch für die Bauteile des Softwarebaukastens gelten soll, werden weiter in In-Process-Server[5] und Out-of-Process-Server unterschieden. Bei Out-of-Process-Server wird der Code unabhängig vom Client ausgeführt, dies kann beispielsweise dazu genutzt werden, um Multithreading zu realisieren, also die sozusagen parallele Ausführung von Teilprogrammen. Bei In-Process-Server handelt es sich um die bereits

[2] Siehe Kapitel 1.2
[3] COM – Component Objekt Model. Firmenspezifischer Name der Firma Microsoft, welcher deren Betriebssystem Windows betrifft
[4] API - Application Programming Interface. Siehe auch folgende Erläuterung im Bericht
[5] Die Fa. Microsoft hat für Serveranwendungen auch das Modewort ActiveX-Exe für Out-of-Process-Server, ActiveX-Dll für In-Process-Server oder allgemein ActiveX-Automation (früher OLE- Automation) kreiert. In der Literatur ist auch manchmal von Code-Komponenten die Rede, womit das Gleiche gemeint ist.

mehrfach erwähnten DLL's oder auch um dynamisch gebundene Bibliotheken. Will ein Client , wie beispielsweise Mathcad oder Excel, Funktionen eines Servers nutzen wird dieser in den Adressraum des Client geladen. Der Server gehört somit zum selben Prozess (Thread) wie der Client. Wird die Client-Anwendung beendet, wird damit auch die Server-Anwendung terminiert. Auf weitere Eigenheiten der unterschiedlichen Servertypen, die beispielsweise auch von der verwendeten Programmiersprache abhängen, wird an dieser Stelle nicht eingegangen und auf nachfolgende Kapitel und weiterführende Fachliteratur verwiesen, wie beispielsweise [1].

Eine Entscheidung, welche Art von Server mit welcher Funktionalität letztlich programmiert oder umgesetzt wird, muss spezifisch für jedes Projekt fallen, da in der Informationstechnologie maßgeschneiderte Lösungen für die unterschiedlichsten Problemstellungen gesucht werden. Unter Umständen kann es sinnvoll sein ein uneinheitliches Konzept zu wählen und beide Varianten einzusetzen.

In Bezug auf den Baustatik-Softwarebaukasten, mit den in Java programmierten Funktionen, der die Basis des Engineering-Desktop darstellt und aus dem sich die einzelnen bereitgestellten Funktionen als Server von den unterschiedlichen Programmen aufrufen lassen sollen, fiel die Entscheidung durchweg auf In-Process-Server. Sollte ein Client, in dessen Adressraum ein geladener Server gerade ausgeführt wird, geschlossen werden, ist es nicht sinnvoll die Funktionalität des Servers aufrecht zu erhalten, da dieser nur Rechenoperationen zur Verfügung stellt und die Ergebnisausgabe anderweitig, bzw. über die Anbindung an die unterschiedlichen Anwendungen realisiert wird.

In Bezug auf die Anbindung der Funktionen an die unterschiedlichen Anwendungen, wäre es bei den Office Anwendungen, die nicht nur in VBA-Macros, sondern teilweise auch in einer VB-Dll umgesetzt wurden denkbar, einen Out-of-Process-Server zu programmieren. Diese Funktionalität stellt hauptsächlich bei komplexen Berechnungen, bei denen die Strandartfunktionalität des Clients, im Hinblick auf die Eingabe von Werten nicht mehr ausreicht, eine Oberfläche zur Verfügung, in der Benutzer Eingaben machen kann. Nun wäre es möglich, ein und dieselbe Oberfläche für verschiedene Office Anwendungen zu verwenden. Durch programmieren eines Out-of-Process-Servers wäre es dann beispielsweise denkbar, einen Server aus Excel heraus zu starten, Werte aus Excel zu übernehmen, Excel zu schließen und aus dem Server, der jetzt in einem eigenen Thread läuft, Ergebniswerte über die Softwarebaukastenfunktionen zu berechnen und als formatierte Ausgabe in ein Word Dokument zu schreiben. Laut den von Microsoft propagierten Möglichkeiten der COM-Technologie, stehen dem Programmierer hier alle denkbaren Möglichkeiten offen, was jedoch oft nicht so reibungslos funktionierte, wie in der Werbung versprochen. Hierbei sei jedoch auf die nächsten Kapitel verwiesen.

Im Projekt wurde die Verwertung von Ergebnissen aus dem Softwarebaukastens auf MS-Excel und MS-Access beschränkt, deshalb hat man sich auch hier für In-Process-Server entschieden, die jedoch aufgrund der gewählten Klassen- und Modularstruktur jederzeit erweiterbar sind. Der Grund hierfür war, dass Excel für statische Berechnungen über wesentlich mehr Funktionalität als beispielsweise MS-Word verfügt. Zudem bietet sich die Möglichkeit per ActivX-Automation Excel-Dateien in Word einzubinden und zu bearbeiten. Somit steht diese Entscheidung mit den Möglichkeiten, welche COM bietet und Microsoft propagiert in Einklang. Der Zweck der COM- und ActiveX-Technologie ist schließlich die Nutzung bereits bestehender Programme mit deren ausgereifter Funktionalität in anderen Anwendungen.

## 1.3 Java[6] und die Java 2 Plattform

Bei der Entwicklung der Programmiersprache Java wurden von verschiedenen Programmiersprachen herausragende Konzepte übernommen, wie Bild 2 zeigt und nachfolgend kurz erläutert wird [2].

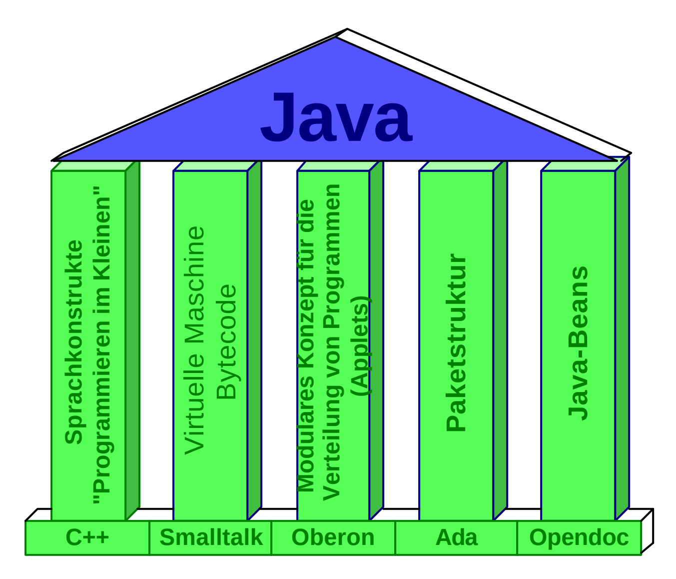



Bild 2. Das Sprachkonzept der Programmiersprache Java

Java hat wie kaum eine andere Sprache ein mehrstufiges Sicherheitskonzept, das die Ausführung von kritischen oder für das System gefährlichen Operationen verhindert.

Die Struktur einer Quellcode-Datei ist im Falle eines Java Programms einfacher als bei einem C++ Programm. Alles was in Java programmiert wird besteht aus Klassen. Während es in C++ noch möglich war, Programme teils prozedural , teils objektorientiert zu entwerfen, lässt Java nur noch Klassen und Objekte zu, es handelt sich also um eine rein objektorientierte Sprache.

Java wurde von Anfang an für die Verteilung von Objekten und Klassen entworfen. Infolge einer umfangreichen Unterstützung durch die Java Klassenbibliothek für Netzwerkverbindungen oder den Zugriff auf verteilte Objekte ist Java optimal für die Client/Server-Programmierung geeignet. Des weiteren wurden durch das Konzept der Applets, die von einem Web-Server auf einen anderen Rechner über das Netz geladen werden können, das Problem der Verteilung bei Software Updates elegant gelöst.

Java ist als plattformübergreifende oder portierbare Programmiersprache konzipiert. Der erstellte Code kann zu Klassendateien mit der Endung .class compiliert werden und sofern auf einem Rechner die Java Virtual Machine installiert ist, auch ausgeführt werden.

Die Java 2 Plattform der Firma Sun [3] stellt mit der Version 1.3 der Entwicklungsumgebung JDK[7] die aktuelle Version der Firma Sun dar. Diese Entwicklungsumgebung wird, anders als

[6] Java ist ein Produkt der Firma Sun, die Kontaktadresse ist unter [3] zu finden. Die aktuelle Version ist die Java 2 Plattform mit der Version 1.3 der Programmiersprache Java

bei Microsoft, unendgeldlich im Netz zum Download bereitgestellt [3]. Da jedoch Dll's eigentlich nur für das Betriebssystem Windows spezifisch sind und mit dem Compiler javac, der im Packet von JDK 1.3 enthalten ist, nicht erstellt werden können, wurde auf ein Entwicklungstool der Firma Microsoft selbst zurückgegriffen, Microsoft Visual J++.

### 1.3.1 Visual J++

Visual J++ ist, wie oben bereits erwähnt ein Produkt der Firma Microsoft. Es besteht aus einer Entwicklungsumgebung, die auf der Java-Version aus dem JDK 1.1 aufbaut. Aus rechtlichen Gründen konnte die Firma Microsoft den Namen Java jedoch nicht beibehalten. Zudem wurden Erweiterungen vorgenommen, die Windows spezifisch sind, vor allem die Anbindung der MFC[8] und nicht in Java enthaltene Standard Schlüsselwörter [4]. Bei der Verwendung des Programms Visual J++ muss dies beachtet werden, falls der Quellcode mehr oder minder portabel bleiben soll. Beim Schreiben des Quellcodes wurde aus Gründen der Portabilität darauf geachtet, dass möglichst keine Klassenbibliotheken und Schlüsselwörter verwendet wurden, die Microsoft spezifisch sind. Allerdings wurde mindestens eine MFC-Klasse benötigt wird, wie in 1.3.1.1 näher erläutert wird.

#### 1.3.1.1 Visual J++ und COM

Nachfolgend wird kurz auf die Entwicklungsumgebung von J++ eingegangen und anhand eines konkreten Beispiels erläutert, wie COM-Klassen erstellt werden. Das Beispiel wird in späteren Kapiteln wieder aufgegriffen, um aufzuzeigen, wie die erstellten COM-Klassen in anderen Programmiersprachen und verschiedenen Programmen genutzt, eingebunden und instanziert werden. Dies verdeutlicht auch die Wirkungsweise der COM-Technologie.

Nach dem Öffnen von Visual J++ erscheint ein Dialogfenster, Bild 3, in dem der Name und Art des Projekts gewählt werden kann.

Nach Bestätigung der Einstellungen wird die Entwicklungsumgebung geöffnet. Das Layout von Visual J++ ist an die Entwicklungsumgebung von Visual Studio angepasst, also ähnlich dem von Visual C++ oder Visual Basic, wie in nachfolgendem Bild zu sehen ist.

[7] JDK – Java Development Kid
[8] MFC – Microsoft Foundation Classes

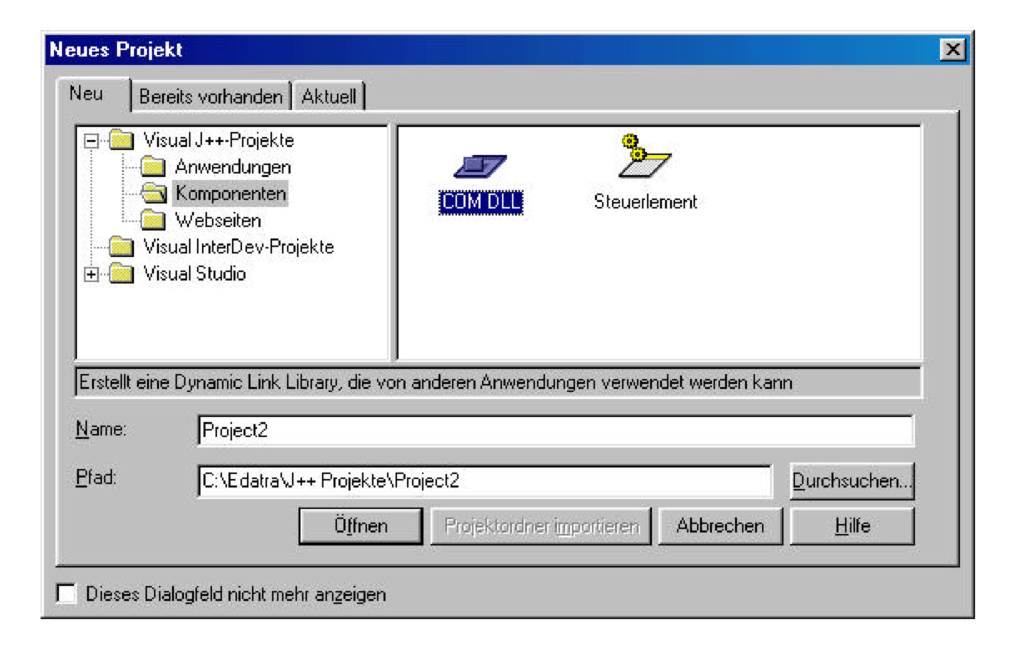



Bild 3. Dialogfenster Neues Projekt der Entwicklungsumgebung Visual J++

Im Projekt-Explorer, auf der rechten Seite von Bild 4, ist die Datei HolzZugNachweis.java markiert. Der Dateiname muss laut Konvention von Java der Gleiche sein, wie der Klassenname in der Datei. Bei der Erstellung einer COM-Dll muss das Packet *com.ms.wfc.app* in die Datei importiert werden, wie in Bild 4 zu sehen ist.

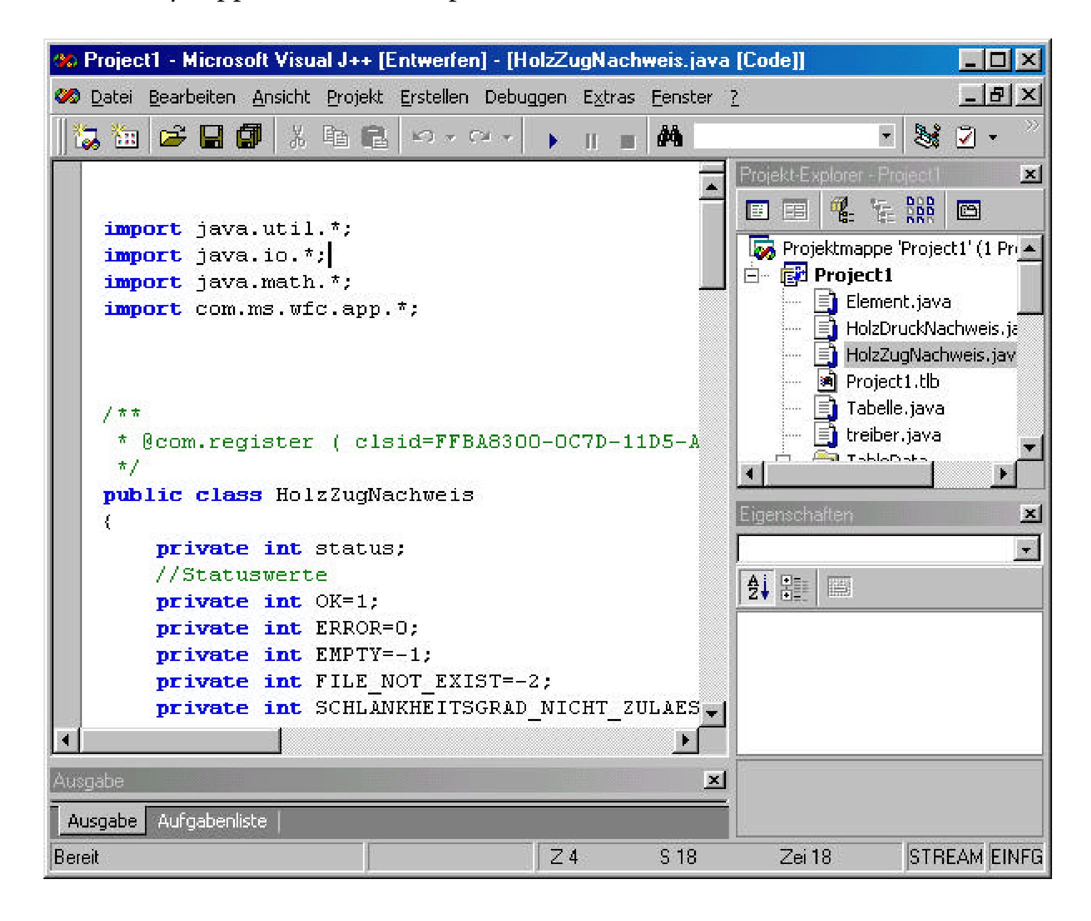



Bild 4. Die Entwicklungsumgebung von Visual J++

Dieses Packet enthält Microsoft-spezifische Klassen, die Zugang zur Windows-Registry erlauben, oder auch Operationen für den Zwischenspeicher oder Windows Anwendungen, wie Threads oder Messaging ermöglichen. Für nähere Informationen wird auf die MSDN-Bibliothek [5] verwiesen.

Im Editor selbst ist eine grün markierte Zeile zu sehen. Diese wird automatisch hinzugefügt, wenn im Dialogfenster Eigenschaften unter dem Tabellenblatt COM-Klassen die entsprechende Klasse als COM-Klasse markiert wird, siehe Bild 5. Innerhalb der Klammer hinter @ com.register sind Informationen über die Registrierung der Klasse als COM-Klasse enthalten, wie beispielsweise den CLSID (Class Identifier) oder die Registrierungsnummer der Bibliothek.

In Bild 4 ist bereits der Code für die erstellte Klasse enthalten. Hierauf wird im entsprechenden Kapitel für die erstellten Funktionen näher eingegangen.

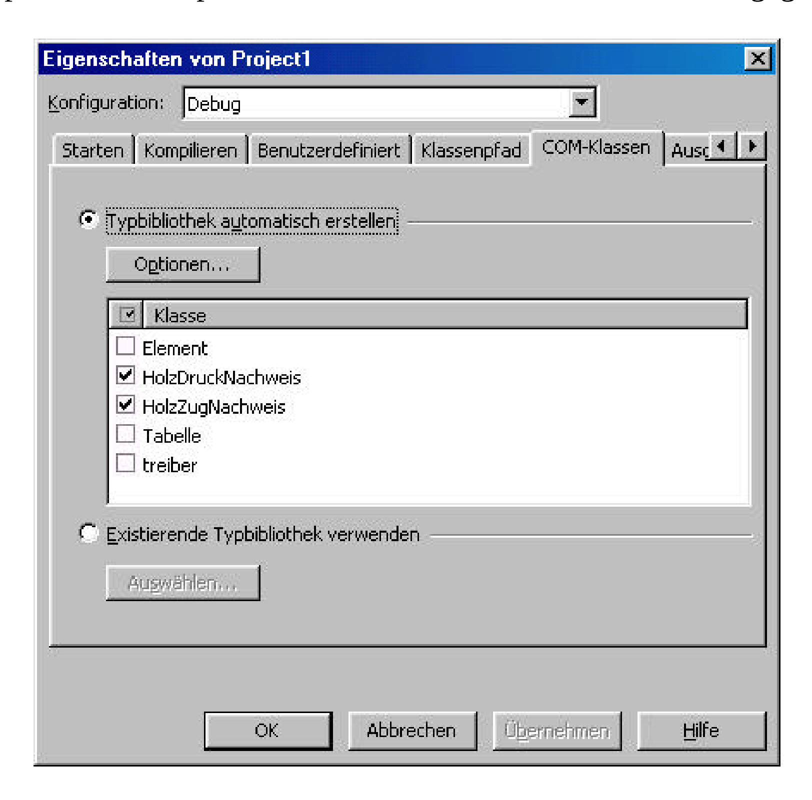



Bild 5. Dialogfenster *Projekt Eigenschaften* – COM-Klassen von Visual J++

Weiterhin kann im Tabellenblatt *Ausgabeformat* der Dateiname gewählt werden, den die Kompilierte Version der Bibliothek erhält oder auch der Verpackungstyp eingestellt werden, falls dies nicht schon bei der Erstellung des Projekts geschah, siehe Bild 6.

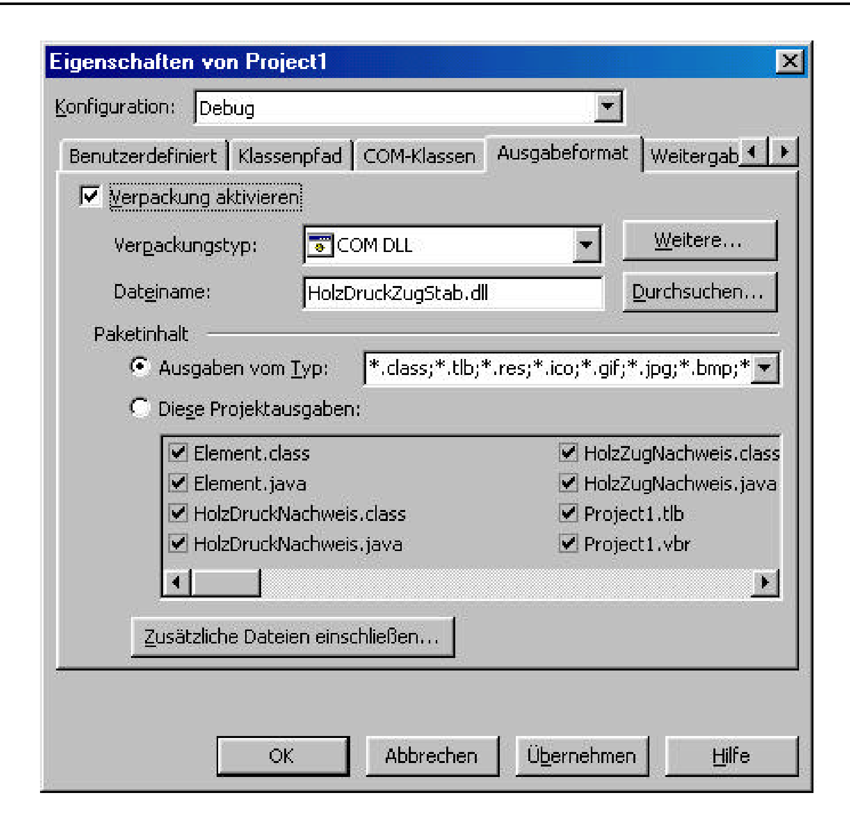



Bild 6. Dialogfenster Projekt Eigenschaften - Ausgabeformat von Visual J++

### 1.3.2 Java, JavaScript und JScript

Auf die beiden Programmiersprachen wird an dieser Stelle kurz eingegangen, da im Rahmen des Projekts Mathcad Blätter entwickelt wurden, um die Formeln und den Berechnungsweg für die Programmierung deutlich zu machen. Um die Blätter möglichst anschaulich zu gestalten wurde überlegt und getestet, inwieweit es sinnvoll ist Skriptobjekte in die Blätter einzubauen. Diese Möglichkeit besteht für die neueren Mathcadversionen beispielsweise mit der Programmiersprache Jscript. Weiterhin stellt die Verwendung von Skriptobjekten eine Variante zur Programmierung in C++ dar. Der Programmierung mit C++ wurde allerdings für die Engineering-Desktop-Umgebung der Vorzug gegeben, da die damit entwickelten Funktionen automatisch allen Arbeitsblättern in Mathcad zur Verfügung stehen. Die Verwendung von Skript-Objekten erfordert hingegen eigene Arbeitsblätter, in denen diese definiert sind.

JavaScript existiert in unterschiedlichen Versionen (JavaScript 1.0, 1.1, 1.2, 1.3, JScript, u.s.w.). Sie ist eine eigenständige Programmiersprache und nicht etwa der Skript-Ableger von Java. JavaScript ist standardisiert. Allerdings haben die unterschiedlichen Browserhersteller ihre Varianten um spezifische, nichtstandardisierte Elemente ergänzt [6]. Somit stellt JScript die Microsoft-spezifische Version der Sprache JavaScript dar.

JavaScript und Java ähneln einander durchaus. Jedoch ist JavaScript nicht so "streng" aufgebaut und verfügt nicht über Javas statische Typen oder die Typprüfung. Dafür hat JavaScript jedoch im wesentlichen die Syntax von Java. Folgende Tabelle stellt die wichtigsten Unterschiede von Java und JavaScript gegenüber.

Von der Sprache JScript existieren mittlerweile insgesamt 7 Versionen, die alle mehr oder minder unterschiedliche Sprachmerkmale oder Features aufweisen.

| JavaScript | Java |
|---|---|
| Der Client interpretiert den Quelltext zur Laufzeit. Der Quelltext wird also nicht compiliert. | Der Quelltext wird zu Klassendateien, mit der Endung .class compiliert und kann über die Java Virtual Machine ausgeführt werden. |
| Objektbasiert. Das Programm nutzt eingebaute Objekte, jedoch keine Klassen oder Vererbung. | Objektorientiert. Objektorientierte Merkmale wie Klassen mit Vererbung stehen zur Verfügung. |
| Der Quelltext wird in HTML-Quelltext integriert und ist dort vollständig (und nachlesbar) enthalten. | Die Java-Applets sind getrennt vom HTML-Dokument gespeichert und werden nur von diesem aufgerufen. |
| Variablen werden nicht explizit deklariert und sind typenlos. | Variablen müssen vor ihrer Verwendung mit Typangabe deklariert sein. |
| "Dynamisches Binden". Die Objektreferenzen werden zur Laufzeit geprüft. | "Statisches Binden". Die Objektreferenzen müssen bereits beim Compilieren existieren. |

Tabelle 1. Die Unterschiede von Java zu JavaScript [6]

Tabelle 2 zeigt, welche Version von JScript mit welcher Host Applikation mitgeliefert wurde. Je nach Rechner, auf dem ein Mathcad-Arbeitsblatt geöffnet wird kann es somit zu Fehlermeldungen kommen.

| **Host Application** | **1.0** | **2.0** | **3.0** | **4.0** | **5.0** | **5.1** | **5.5** |
|---|---|---|---|---|---|---|---|
| Microsoft Internet Explorer 3.0 | **x** | | | | | | |
| Microsoft Internet Information Server 1.0 | | **x** | | | | | |
| Microsoft Internet Explorer 4.0 | | | **x** | | | | |
| Microsoft Internet Information Server 4.0 | | | **x** | | | | |
| Microsoft Windows Scripting Host 1.0 | | | **x** | | | | |
| Microsoft Visual Studio 6.0 | | | | **x** | | | |
| Microsoft Internet Explorer 5.0 | | | | | **x** | | |
| Microsoft Internet Information Services 5.0 | | | | | | **x** | |
| Microsoft Windows 2000 | | | | | | **x** | |
| Microsoft Internet Explorer 5.5 | | | | | | | **x** |

Tabelle 2. Lieferumfang der Sprache JScript in Bezug auf die Version des Internetexplorers [9]

Da weder in der Online Hilfe von Mathcad, noch im Benutzerhandbuch oder der Homepage von Mathsoft befriedigende Informationen über die Verwendung von Skriptobjekten bereitgestellt werden, entstand im Rahmen des Projekts ein kleines Tutorial für die Implementierung der Objekte in Mathcad Blättern, das im Folgenden aufgeführt ist.

[9] Die Tabelle stammt aus der Dokumentation zu JScript. Die Dokumentation kann von der Microsoft Homepage heruntergeladen werden: http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001169

#### 1.3.2.1 Verwenden der Script Objekte in Mathcad




Um die Objekte einzufügen wird über den Menübefehl *Einfügen|Komponente* ein Assistent gestartet, der durch den Prozess begleitet.

Bild 7. Das Menü *Einfügen* von Mathcad 2000

Im Assistent Auswahl *Skriptobjekt* wählen.

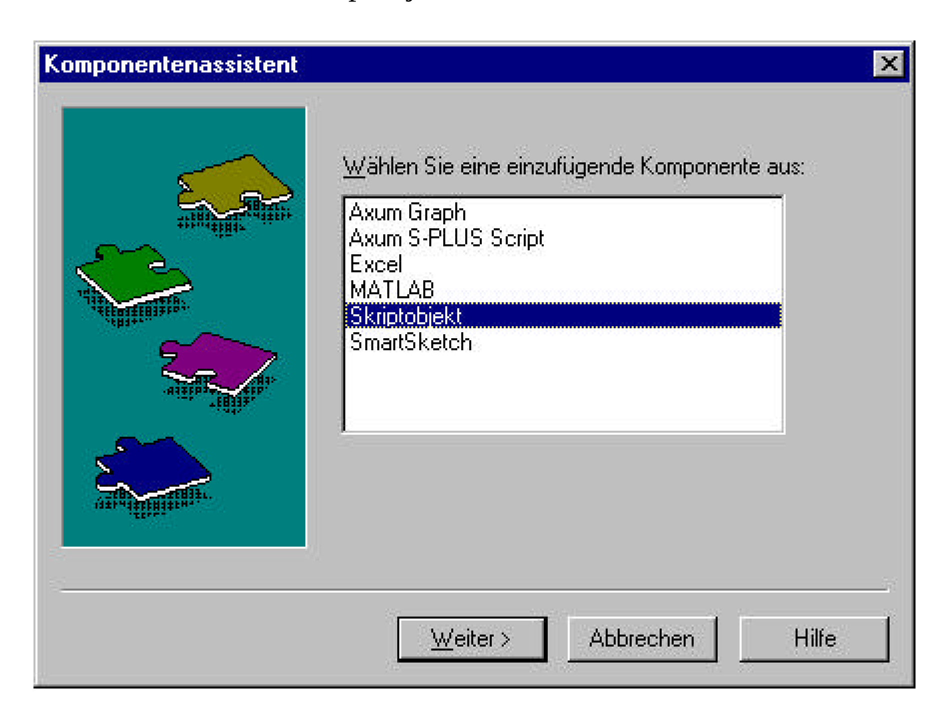



Bild 8. Der Komponentenassistent von Mathcad 2000

Nach dem Klicken auf *Weiter* erscheint ein weiteres Fenster mit einer langen Liste an verfügbaren Objekten. Bei den Versuchen innerhalb des Projekts wurden lediglich Steuerelemente der *Microsoft Forms 2.0* Objektbibliothek ( bis 'M' weiter scrollen) getestet. Dies sind die gleichen Objekte (Steuerelemente), wie sie auch in den Office-Anwendungen in der Programmiersprache VBA zur Verfügung stehen. Die Steuerelemente sind eine etwas abgespeckte Version der Steuerelemente von Visual Basic. Das heißt es stehen nicht alle Eigenschaften zur Verfügung wie in VB.

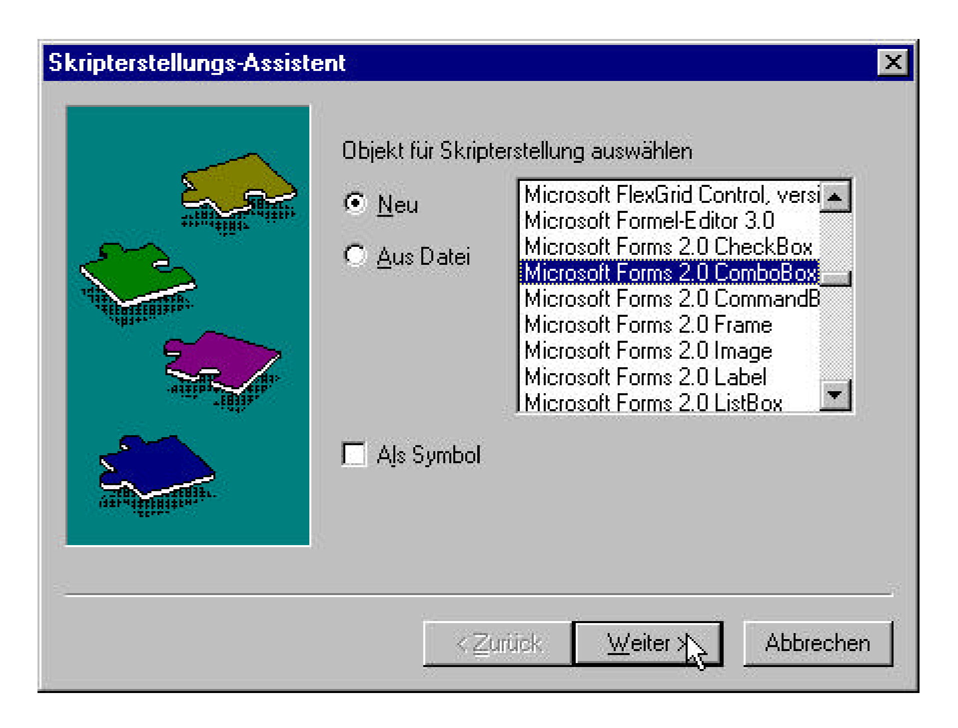



Bild 9. Der Skripterstellungs-Assistent (Stufe 1) von Mathcad 2000

Bei den Tests im Projekt wurden wieder exemplarisch einige wenige Steuerelemente, wie Combo-Boxen zur Auswahl von Elementen und Textboxen zum anzeigen von Text verwendet. Unglücklicherweise konnte jedoch beispielsweise schon bei Label-Feldern auf verschiedene Eigenschaften der Steuerelemente nicht zugegriffen werden. Das Problem lässt sich möglicherweise lösen, worauf aber im Projekt aus Zeitgründen verzichtet wurde.

Im Folgenden kann dann eine Script-Sprache ausgewählt werden. Um eine gewisse Konsistenz innerhalb des Projekts zu erhalten, wurde bei den verwendeten Steuerelementen JScript verwendet.

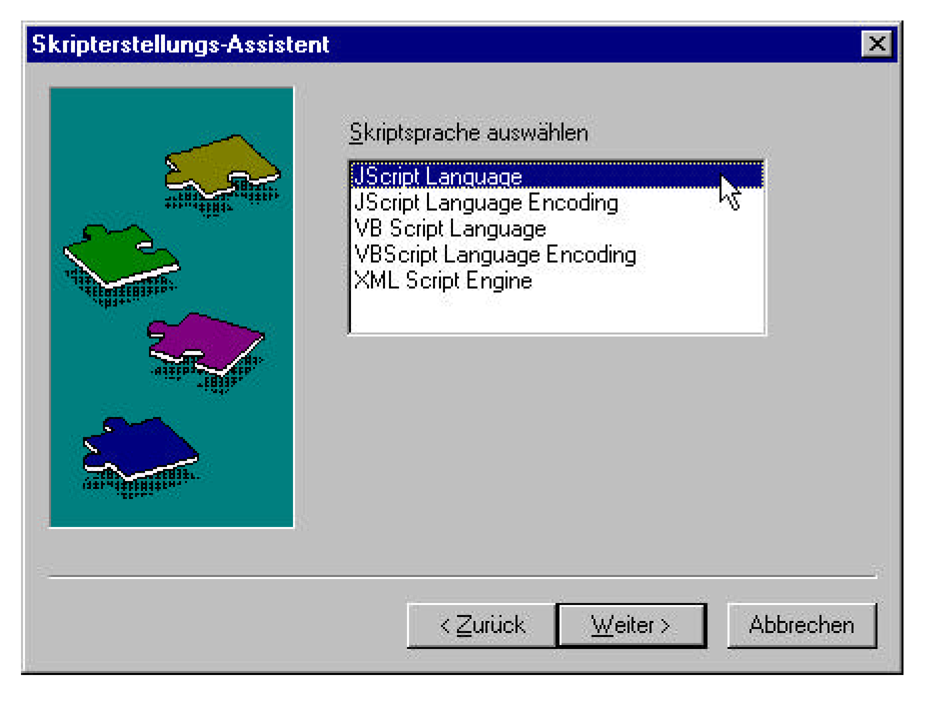



Bild 10. Der Skripterstellungs-Assistent (Stufe 2) von Mathcad 2000

Eine Dokumentation zu VBScript und JScript kann von der Microsoft Homepage heruntergeladen werden[10]. Aus dieser Dokumentation stammt auch Tabelle 2. Mathcad selbst stellt im Ordner „...\Programme\MathSoft\Samples\Scripted“ zwei Blätter zur Verfügung, auf denen verschiedene Steuerelemente verwendet werden. Die Beispiele sind in VBScript[11] programmiert. Weiterhin sind in [7] in mehreren Arbeitsblättern weitere Beispiele für die Verwendung von Skriptobjekten in Mathcad, programmiert in Jscript zu finden.

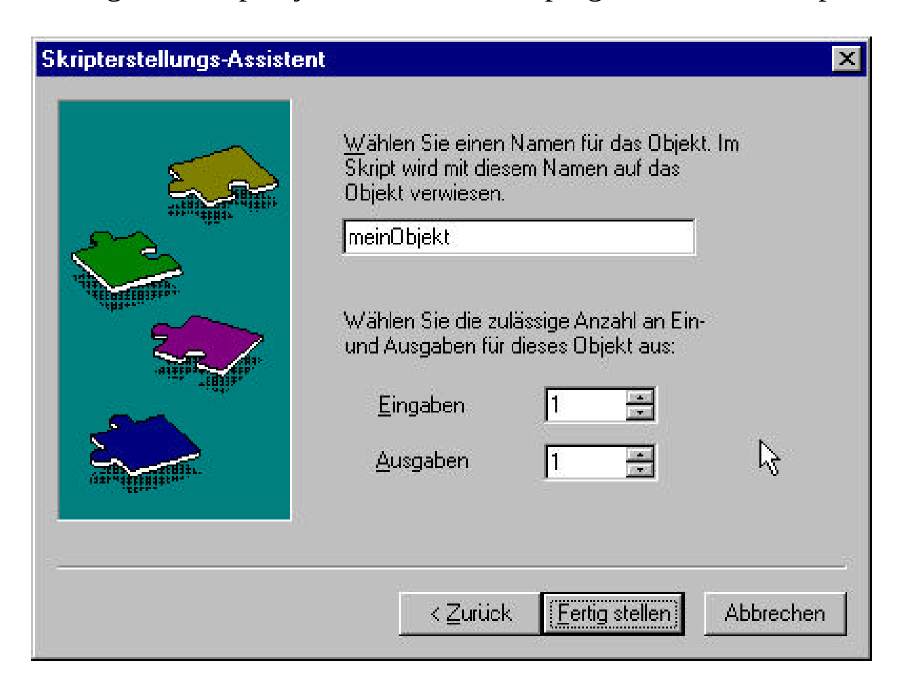



Bild 11. Der Skripterstellungs-Assistent (Stufe 2) von Mathcad 2000

Nachdem die Scriptsprache gewählt ist können im nächsten Fenster (siehe Bild 11) weitere Optionen eingestellt werden. In der oberen Textbox wird ein Name für das Objekt eingegeben. Über diesen wird das Objekt später im Code angesprochen. Unten kann die Anzahl der Ein- und Ausgabevariablen eingestellt werden. In beiden Fällen sind je vier Ein- bzw. Ausgaben möglich. Dies können beispielsweise ganz normale Variablen im Mathcad Blatt sein, aber auch Matrizen, was die Menge an Daten, die übergeben werden kann vervielfacht.

Nach dem Fertigstellen ist die Combobox im Arbeitsblatt sichtbar.

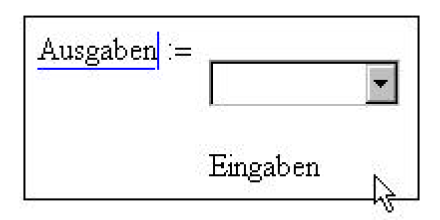



Bild 12. Combobox in Mathcad

[10] URL: http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001169

[11] VBScript ist eine Scriptsprache, die vom Client interpretiert wird. Wie in VBA wird der Quelltext zur Laufzeit interpretiert. VBScript ist eine Teilmenge von VB und Microsofts Antwort auf JavaScript. Mehr Informationen sind unter http://www.microsoft.com/scripting/ zu finden.

Bei *Ausgaben* im oberen Bild kann, wie gewöhnlich eine Variable deklariert werden. Hierbei ist jedoch zu beachten, dass dieser dann nicht der String (Zeichenfolge) oder Zahl zugewiesen wird, der in der Box ausgewählt wurde, sondern eine Wert, der über den Code der Variablen zugewiesen wird.

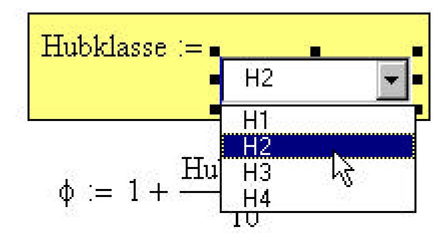



Bild 13. Fertiggestellte Combobox in Mathcad

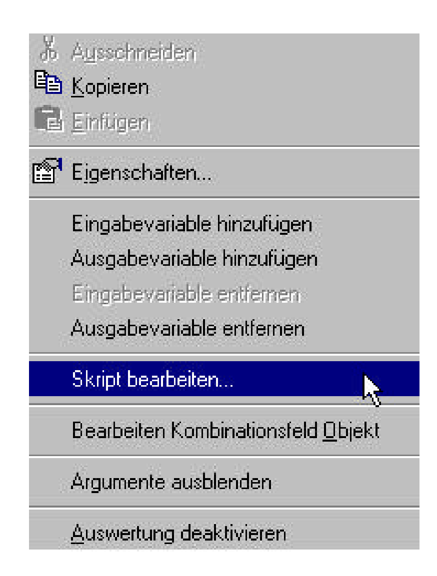



Bild 14. Popup Menü für Skriptobjekte in Mathcad

Die Bearbeitung des Code findet im Code Editor statt. Um diesen aufzurufen, bedarf es eines rechten Mausklicks auf dem Steuerelement. Damit wird das unten gezeigte Popup Menü aufgerufen. Nach der Auswahl von *Skript bearbeiten*, wird der Editor geladen

Im Script Editor sind nach dem Laden bereits drei Funktionsköpfe vorhanden. *Start* wird ausgeführt, wenn das Mathcadblatt geladen wird. *Exec*, wenn Änderungen an den Eingabevariablen stattfinden, was auch durch die beiden Parameter in Klammer (Inputs, Outputs) deutlich wird, über die Werte übergeben werden. *Stop* schließlich wird ausgeführt, wenn das Blatt aus dem Arbeitsspeicher geladen wird.

Skripteditor - [Text]

Datei Bearbeiten Ansicht ?

```
var dataval = 0


function HubEvent::Start()
{
  // TODO: Add code here
//      Hub.ListIndex = (dataval)

}

function HubEvent::Exec(Inputs,Outputs)
{
  // TODO: Add code here
        dataval=Hub.ListIndex;

        if  (dataval== -1)
        {
        Hub.AddItem("H1");
        Hub.AddItem("H2");
```

Bereit

Bild 15. Der Skripteditor von Mathcad

Folgend ist der Code für die obige Combobox aufgeführt:

```
// Deklaration einer Variablen. Man beachte: var mit nachgestelltem Variablennamen. Die
// Zuweisung eines Typs erfolgt nicht mit double oder int. Hierzu wird einfach ein Integer
// oder Double Wert zugewiesen.
var dataval = 0
function HubEvent::Start()
{
// TODO: Add code here
}

function HubEvent::Exec(Inputs,Outputs)
{
// Abfragen der ListIndex Eigenschaft der Combobox. Falls diese -1 liefert, wird die Box über
// AddItem mit Strings gefüllt. Die ListIndex-Eigenschaft hat den Wert - 1, wenn kein
// Element ausgewählt ist oder wenn ein Benutzer eine Auswahl in einem Kombinationsfeld
// eingibt (Style = 0 oder 1), anstatt ein vorhandenes Element aus der Liste auszuwählen.
        dataval=Hub.ListIndex;
        if  (datava l= = -1)
                {
        Hub.AddItem("H1");
                Hub.AddItem("H2");
                Hub.AddItem("H3");
                Hub.AddItem("H4");
                dataval=0;
```

```
        }
// Nachdem ein Element ausgewählt wurde, wird diesem ein Wert zugewiesen. Beachte hier.
// Outputs stellt hier selbst ein Objekt dar, deshalb der Zugriff auf die Value Eigenschaft
// durch einen Punkt getrennt mit vorangestelltem Objekt.

        switch (Hub.ListIndex)
        {
                case 0:
                Outputs(0).Value(0) = 1;
                break;
                case 1:
                Outputs(0).Value(0) = 2;
                break;
                case 2:
                Outputs(0).Value(0) = 3;
                break;

                case 3:
                Outputs(0).Value(0) = 4;
                break;
                }
        }
// Eigentlich fehlt noch eine Default Anweisung innerhalb der Switch Anweisung. Auch
// wurden noch weitere Fehler nicht abgefangen, was jedoch für diese Anwendung durchaus
// genügt, da nach dem ersten Speichern des Blatts mit der Box die Einträge in der Box
// scheinbar erhalten bleiben.

function HubEvent::Stop() {
  // TODO: Add code here
}
```

Das folgende Beispiel gilt für eine Textbox, die lediglich anzeigt, ob ein Nachweis eingehalten wurde oder nicht.

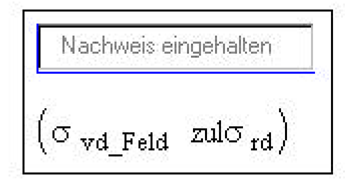



Bild 16. Textbox mit Übergabeparametern in Mathcad

Wie an den Eingabevariablen zu sehen ist, werden hier zwei Werte übergeben. Der folgende Code zeigt die weitere Auswertung der beiden Variablen. In diesem Fall werden keine Ausgabewerte benötigt.

```
function txt_SigmaVEvent::Start()
{
  txt_SigmaV.Enabled = 0                    // Enabled: Werte 0 oder 1. 0 bedeutet, dass der Benutzer
```

```
                                        // keine Eingaben machen kann. Macht die Textbox
                                        // praktisch zu einem Labelfeld.
}
function txt_SigmaVEvent::Exec(Inputs,Outputs)
{
  var SigmaV=0.
  var fyk=0.
  SigmaV=Inputs(0).Value
  fyk=Inputs(1).Value
    if (SigmaV<=fyk) {
           txt_SigmaV.text="Nachweis eingehalten";
    }
    else if  (SigmaV>fyk) {
           txt_SigmaV.Text = "Nachweis nicht erfüllt";
    }
}
```

Weiterhin können auch eigene Funktionen geschrieben werden, bzw. Objekte erstellt werden, die notwendige Berechnungen ausführen.

```
function ScriptObjEvent::Exec(Inputs,Outputs)
{
var a=0.0;
var a= Inputs(0).Value;
var oh= test(a);
ScriptObj.Text=oh;                  // oder: ScriptObj.Text=test(a)
}

function test(myValue) {
var soso=0.0;
soso=myValue*myValue;
return soso;
}
```

Eine weitere Möglichkeit ist über einen Konstruktor ein Objekt zu erzeugen. Dieses kann dann wie gewöhnlich initialisiert werden, was folgendes Beispiel zeigt:

```
function ScriptObjEvent::Exec(Inputs,Outputs) {
var a=0.0;
var a= Inputs(0).Value;
var oh= new test(a);                // Initialisierung des Objeks über den Aufruf des Konstruktors
ScriptObj.Text=oh.soso;                    // Zugriff auf die Eigenschaft, durch einen Punkt getrennt vom
Namen.
}
// Konstruktor des Objekt
function test(myValue){
this.soso=myValue*myValue;
}
```

Programmlisting 1.

Wie zu erkennen ist, werden, anders als in Java selbst, Objekte deklariert ohne die reservierten Wörter class, public usw. zu verwenden, da diese nicht in die Sprache JScript integriert wurden und diese Deklarationen somit auch in der neuesten Version 5.5 von JScript nicht möglich sind. Diese Sprachmerkmale sind jedoch geplant, wie kleine folgende Tabelle zeigt, die ebenfalls aus der Microsoft-Dokumentation zu JScript stammt.
Es ist also durchaus möglich, die Skriptobjekte in Mathcad für aufwendigere Berechnungen heranzuziehen, ob dies praktisch jedoch sinnvoll ist, bleibt von Fall zu Fall zu unterscheiden.

| | | | | |
|---|---|---|---|---|
| break | Delete | function | return | typeof |
| Case | Do | if | switch | var |
| Catch | else | in | this | void |
| continue | false | instanceof | throw | while |
| debugger | finally | new | true | with |
| Default | for | null | try | |

Tabelle 3. Reservierte Wörter in Jscript 5.5

| | | | | |
|---|---|---|---|---|
| abstract | Double | goto | native | static |
| boolean | Enum | implements | package | super |
| Byte | Export | import | private | synchronized |
| Char | Extends | int | protected | throws |
| class | Final | interface | public | transient |
| const | Float | long | short | volatile |

Tabelle 4. Zukünftig geplante Wörter in JScript

Die Untermenge der Sprache Java wird also immer leistungsfähiger. Falls alle unten genannten Features in die Sprache integriert werden, stellen die Skriptobjekte mit der Sprache JScript innerhalb von Mathcad Blättern eine sehr interessant Option dar, da auch sehr aufwendige Berechnungen aus einem Arbeitsblatt in Scriptobjekte verlagert werden könnten. Da die vier Eingabewerte für die Skriptobjekte auch aus Matrizen bestehen können, die man sich innerhalb eines Blattes selbst generieren und übergeben kann, stellt auch dies nicht wirklich ein Hindernis dar. Da die Programmiermöglichkeiten in Mathcad selbst, einigermaßen beschränkt sind, stellt die Verwendung der Objekte für zukünftige Versionen von Mathcad und JScript eine durchaus interessante Möglichkeit dar.

#### 1.3.2.2 Automatisierung von COM Komponenten über Script Objekte

Die Möglichkeit der Atomatisierung von COM Komponenten im Script Code von Script Objekten basiert wiederum auf der COM Technologie.

Da die Automatisierung als eine Technik definiert ist, ist sie zunächst unabhängig von der Programmiersprache. Es ist möglich, diese Technologie aus C++, Basic, Skript- und anderen Programmiersprachen zu nutzen. Microsoft erwähnt in diesem Zusammenhang die Werkzeuge Visual C++, Visual Basic for Applications (VBA) und Visual Basic Scripting (VBScript). Für nachfolgende Erläuterungen und Beispiele für die Benutzung der Objekte wird daher auf die Scriptsprache von Visual Basic zurückgegriffen.

Die Umsetzung der Automatisierung in eigenen Applikationen ist leider nicht trivial. Deswegen wird es nötig auf existierende Entwicklungswerkzeuge zurückzugreifen, welche den Grossteil der Arbeit übernehmen und den Programmierer nicht mit den internen Details konfrontieren,.

Von Microsoft wird dafür die Sprache C++ empfohlen. Diese Programmiersprache wird bei der Entwicklung der meisten, sich auf dem Markt befindenden, Standardprogramme verwendet. Die von Microsoft selbst angebotene Entwicklungsumgebung Visual C++ ist sehr ausgereift und bietet eine hervorragende Unterstützung sämtlicher Windows Technologien, wozu auch die Automatisierung gehört. Zusätzlich existiert auf dem Markt eine Fülle von Literatur, die sich mit diesem Thema auseinandersetzt.

Das Zusammenspiel bei der Automatisierung findet zwischen zwei Objektarten statt. Wie auch an vielen anderen Stellen werden die Dienste oder Funktionen von einem Automatisierungsserver zur Verfügung gestellt. Auf diese können dann verschiedene Automatisierungsclients zugreifen. Der Server ist als ein Objekt zu verstehen, der Methoden und Eigenschaften (Parameter) beinhaltet. Diese können von den Clients beliebig genutzt werden. Dabei ist es unerheblich, in welcher Programmiersprache die Clients geschrieben sind. Es muss lediglich eine Möglichkeit bereitgestellt werden, über die das Objekt initialisiert werden kann. Folgende Grafik stellt die Kommunikation bei der Automatisierung dar.

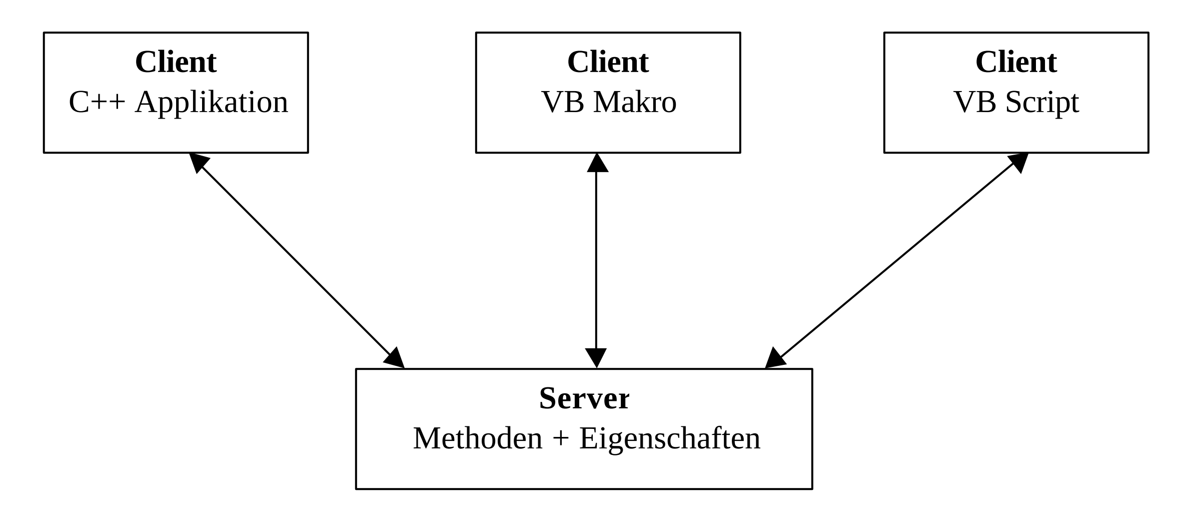



Bild 17. Kommunikation bei der Automatisierung

Damit der Server im System bekannt wird und damit von den Clients genutzt werden kann muss er in der Windows Registrierung eingetragen werden. Das Betriebssystem sorgt automatisch dafür, das jede Anfrage an das richtige Objekt weitergeleitet wird. Hierfür gibt es zwei Möglichkeiten:

- Läuft bereits die Serverapplikation auf dem Rechner, wird diese Instanz des Programms benutzt, um die Anfrage zu bearbeiten.
- Andernfalls wird der Server automatisch gestartet und die Anfrage bearbeitet.

Der Benutzer merkt in beiden Fällen nicht, dass seine Anfrage von einer fremden Applikation behandelt wurde, da dies nichts auf dem Bildschirm angezeigt wird.

#### 1.3.2.2.1 Erstellen eines Automatisierungsservers

Das Herzstück der Automatisierung ist der Server. Im folgenden wird erläutert, wie eine einfache Serverapplikation in C++ mit dem Microsoft Visual Studio C++ 6.0 erstellt werden kann. Dieses Werkzeug bietet sehr gute Unterstützung bei der Entwicklung derartiger Objekte.

Der Automatisierungsserver ist in diesem Fall ein Windows Programm, dass mit Hilfe der MFC Klassenbibliothek erstellt wird[12]. Aus diesem Grund lässt es sich auch wie ein normales Programm konzipieren und entwickeln. Der Programmierer behält die freie Hand bei der Gestaltung seiner Softwarearchitektur in Bezug auf die Klassenbildung und dergleichen. Lediglich eine Objektschnittstelle muss definiert werden, damit die Clients auf die Funktionalität zugreifen können. Die einzelnen Schritte der Erstellung sollen unten aufgezeigt werden.

[12] Es wird an der Stelle empfohlen, das Thema der Erstellung von Windows Applikation, insbesondere die Benutzung von MFC mittels geeigneter Literatur zu vertiefen.

1. Im ersten Schritt muss ein Projekt (Typ *MFC-Anwendungs-Assistent (exe)*) erstellt werden.

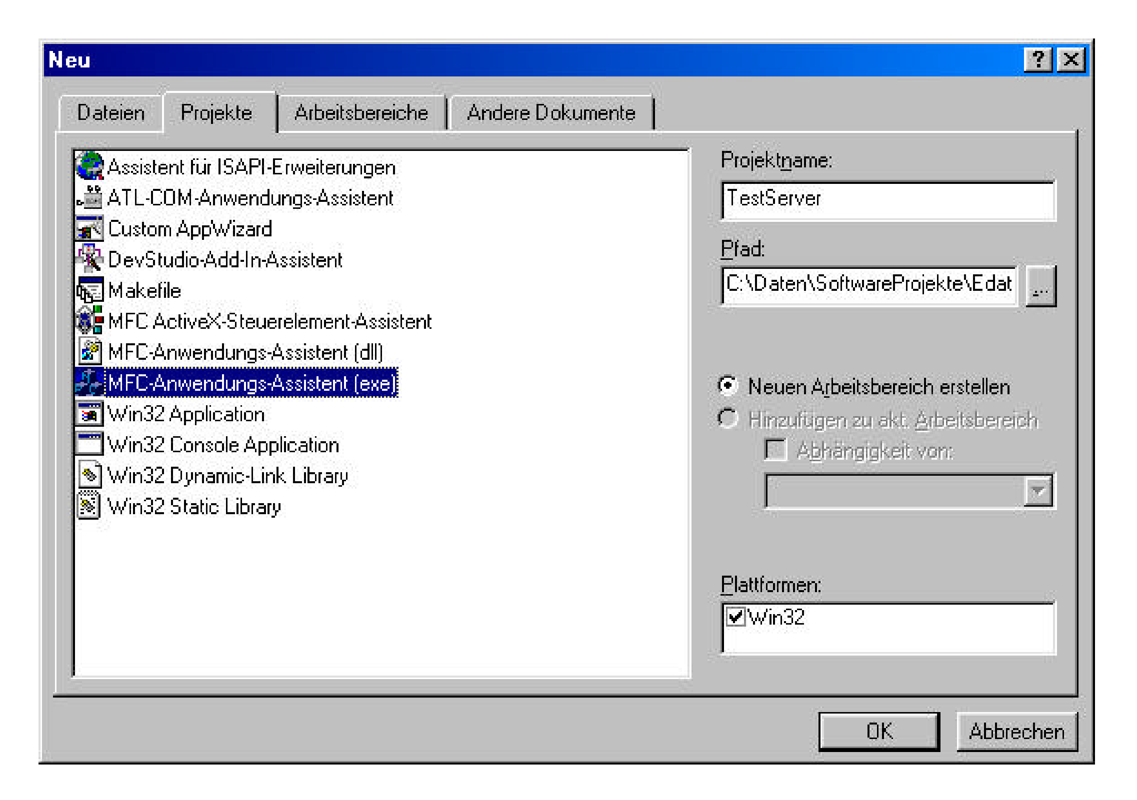



Bild 18. Dialogfenster für die Erstellung von Anwendungen von Visual C++

2. Damit der Server kein Feedback auf dem Bildschirm während der Bearbeitung ausgibt muss in der folgenden Dialogbox (Schritt 1 der Erstellung) die Option *Einzelnes Dokument (SDI)* gewählt werden.

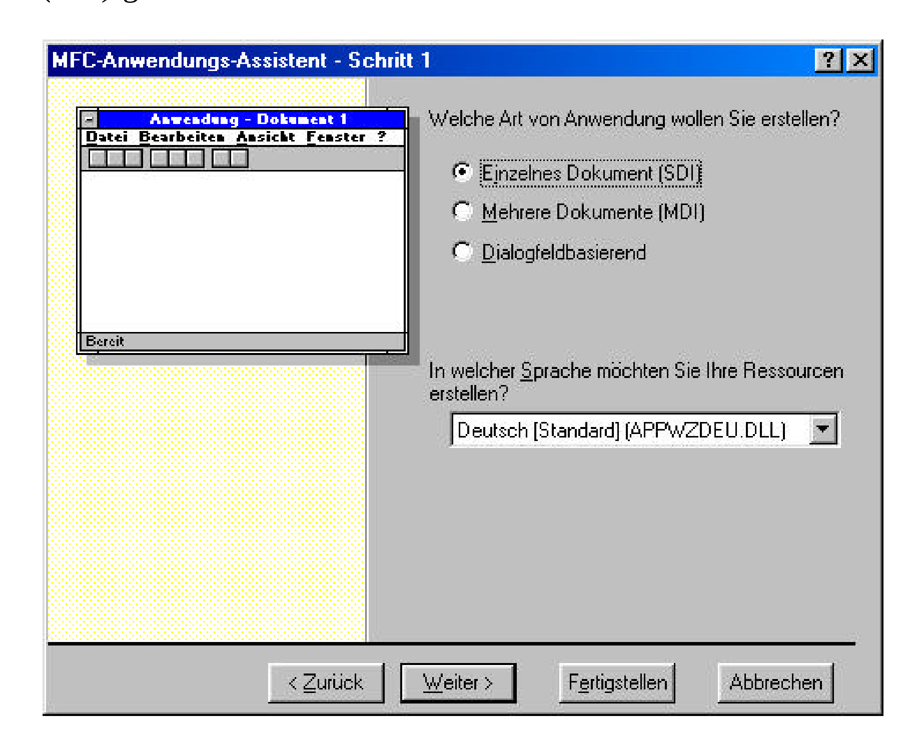



Bild 19. MFC Anwendungsassistent – Schritt 1

3. In einem der nächsten Dialoge (Schritt 3 der Erstellung) muss die Option *Automatisierung* aktiviert werden

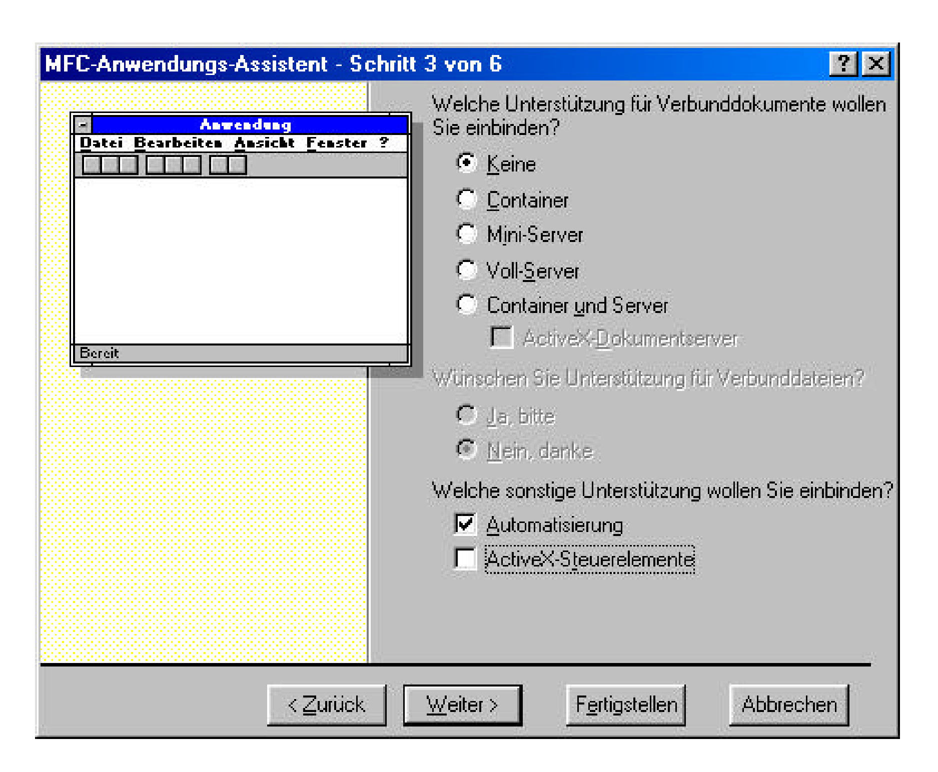



Bild 20. MFC Anwendungsassistent – Schritt 3

4. Im darauffolgendem Dialog (Schritt 4 der Erstellung) muss der Button *WeitereOptionen* gewählt werden. Es wird ein weiterer Dialog eingeblendet, in dem der Name der Objekts für die Automatisierung vergeben werden kann (im Feld *Dateityp-ID*). Im Beispiel lautet der Name „TestServer.Object“.

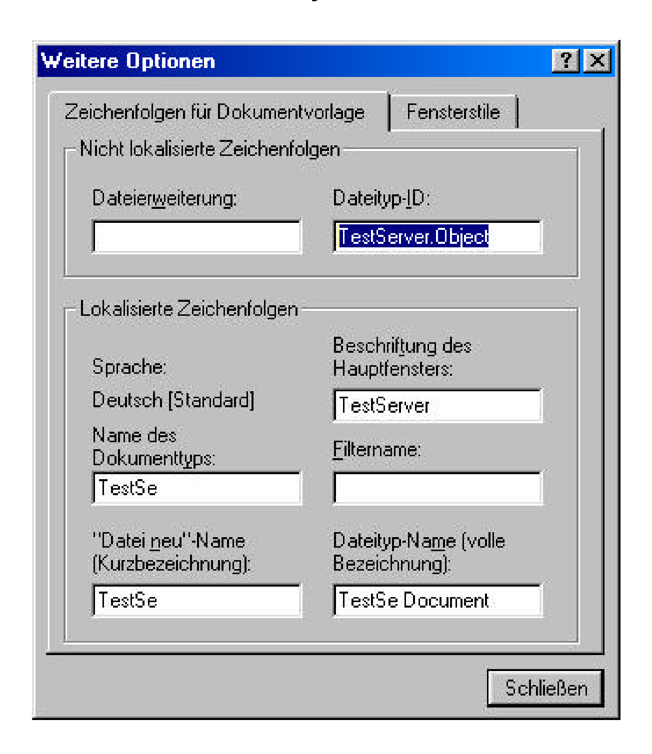



Bild 21. MFC Anwendungsassistent – Schritt 4, Weitere Optionen

5. Nach dem Fertigstellen des Projekts erzeugt Visual Studio automatisch alle notwendigen Klassen, die zum Benutzen der Automatisierung nötig sind. Das Eingreifen des Entwicklers ist an dieser Stelle nicht nötig.

6. Um den Server mit Funktionalität auszustatten, müssen noch die Objektmethoden und/oder Eigenschaften definiert werden. Auch hier bietet Visual Studio entsprechende Hilfe an. Mit dem Klassen-Assistenten lässt sich diese Aufgabe mit nur wenigen Mausklicks erledigen.

Der Assistent wird aus dem Hauptmenü *Ansicht→Klassen-Assistent* aufgerufen. In der Dialogbox unter *Automatisierung* sind die notwendigen Werkzeuge zu finden. Als Klassenname soll hier zunächst die Document-Klasse gewählt werden (Beispiel *CTestServerDoc*). Von allen erzeugten Klassen lässt sich nur diese für die Schnittstelle der Automatisierung verwenden.

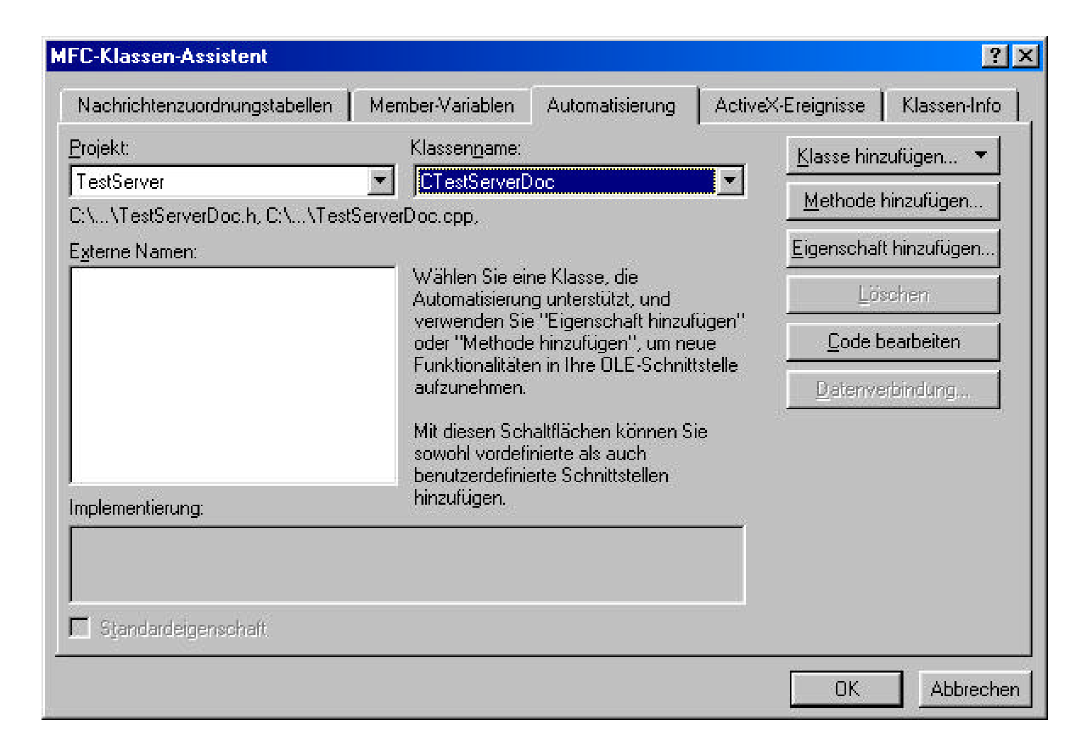



Bild 22. MFC Klassenassistent der Entwicklungsumgebung von Visual C++

7. Mit dem Button *Methode hinzufügen* kann man in dem Objekt beliebige Methoden definieren, die später den Clients zur Verfügung stehen. Die Definition erfolgt auch in einer Dialogbox.

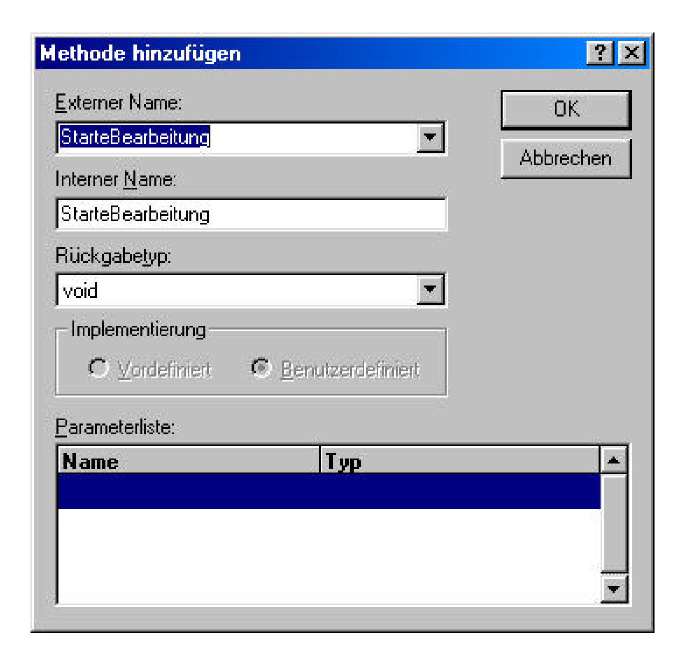



Bild 23. Dialogbox Methode hinzufügen der Entwicklungsumgebung von Visual C++

In diesem Dialog muss für die Methode ein sog. „Externer Name“ vergeben werden. Unter diesem lässt sich die Methode im Client ansprechen. Intern wird sinngemäß der gleiche Name vergeben (eine Abweichung ist jedoch durchaus zulässig). Außer dem Namen werden noch der Rückgabewert und evtl. Parameter festgelegt (im Beispiel wurde auf Parameter verzichtet).

8. Auf ähnlicher Weise können mit dem Button *Eigenschaft hinzufügen* auch Attribute (Variablen) in dem Objekt erzeugt werden.

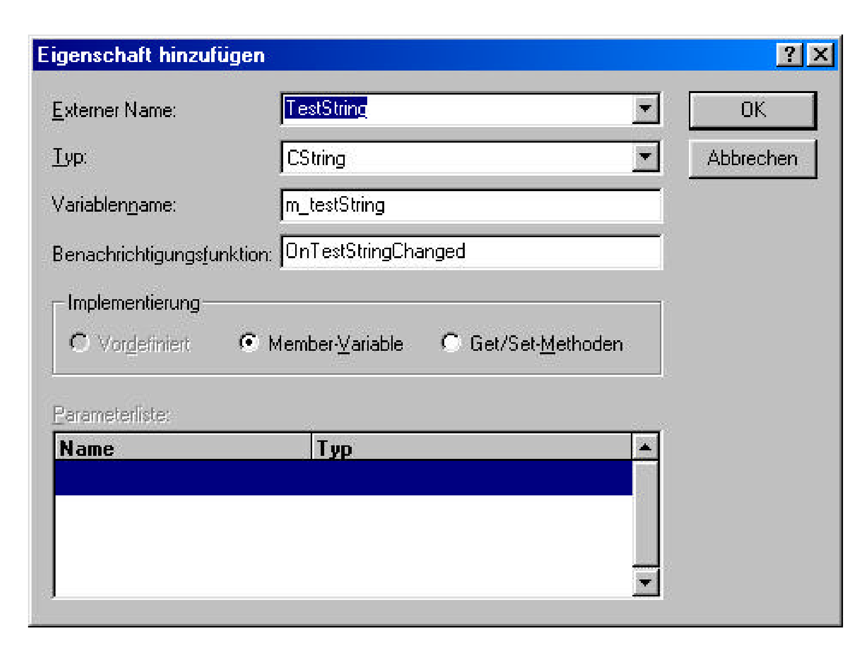



Bild 24. Dialogbox Eigenschaft hinzufügen der Entwicklungsumgebung von Visual C++

Auch hier wird ein externer und interner Name vergeben. Jede variable lässt sich entweder direkt oder über Get/Set-Methoden ansprechen, was in der Dialogbox auch festgelegt werden kann.

9. Der Klassen-Assistent fügt der Document-Klasse die entsprechenden Daten hinzu, so dass dort jetzt die definierten Variablen (Membervariablen der Klasse), sowie Methoden zu finden sind. Zusätzlich wird automatisch die Automatisierungsschnittstelle um die externen Namen ereitert.

Durch Implementierung der definierten Methoden (vom Klassen-Assistenten werden nur leere Rümpfe erzeugt) lässt sich das Objekt mit gewünschter Logik ausstatten und ist nach der Compilierung sofort lauffähig.
Um das Objekt aus einem Client ansprechen zu können, muss die Applikation einmal gestartet werden damit der Server in der Windows Registrierung eingetragen werden kann. Diese Aufgabe kann später auch ein Installationsprogramm übernehmen, das die notwendigen Schlüssel von Visual Studio erzeugt werden (sie sind in der Datei „TestServer.reg“ abgelegt).

#### 1.3.2.2.2 Erstellen eines Automatisierungsclients

Das erstellen eines Clients soll am Beispiel eines Visual Basic Skriptes dargestellt werden. Daran erkennt man, wie einfach es ist den zuvor erzeugten Server zu benutzen.

```
Zeile 1:    Set MyServer = CreateObject( "TestServer.Object" )
Zeile 2:    TestServer.TestString = "TestEingabe 10 22 35"
Zeile 3:    TestServer.StarteBearbeitung()
Zeile 4:    Ergebnis = TestServer.TestString
Zeile 5:    MsgBox ( Ergebnis )
```

Programmlisting 2.

Die Objektvariable (Eigenschaft) wird zunächst mit den Eingaben initialisiert, dann die Bearbeitung gestartet. Zuletzt können die Ergebnisse an der gleichen Stelle abzulesen. Denkbar wäre auch eine Kombination aus den Eingangs- und Ergebnisvariablen oder gar ein Funktionsrückgabewert. Der Kreativität sind hier kaum Grenzen gesetzt. Ein ähnlicher Code lässt sich auch als ein VBMakro verwenden und aus jeder Standard-Officeanwendung aufrufen.

## 1.4 Visual Basic

VB entstand aus der Idee, den DOS-Basic[13] Entwicklern einen Wechsel in die Windowsprogrammierung zu erleichtern. Weiterhin sollte ein leichterer und schnellerer Einstieg für Individual- beziehungsweise Gelegenheitsprogrammierer in das Erstellen von Software mit Windowsoberfläche gegeben werden.

Der Schwerpunkt liegt seit Version 4.0, bei der es sich prinzipiell um ein komplett neues Entwicklungssystem handelt, bei Client-Server und Web-Anbindung. Außerdem liegt VB in allen Microsoft Office Produkten als VBA, also Visual Basic for Applications vor. Dies ermöglicht es, Module direkt in Standartanwendungen zu integrieren.

[13] Basic: Beginners All Purpose Symbolic Instruction Code, Programmiersprache, die sich Anfang der siebziger Jahre entwickelte

Visual Basic ist heute die unter Windows am häufigsten eingesetzte Programmiersprache. Mit der im Projekt verwendeten Version 6 lassen sich Tools jeglicher Art, Multithreading fähige Komponenten, Client-Server-Systeme und Datenbank- bis hin zu Internetanwendungen (Server- sowie Clientseitig) entwickeln. Auch die ADO[14]- oder DCOM[15]-Technologie sind mit VB möglich und dies in einer wesentlich kürzeren Einarbeitungszeit als in den Programmiersprachen C oder C++ .

Die Entwicklung der Programmiersprache VB in den letzten Versionen zeigt, dass von Mal zu Mal immer mehr objektorientierte Aspekte in die Sprache aufgenommen wurden. Sogar dynamische Bindung[16] über polymorphe Objektvariablen ist möglich. Was hingegen fehlt ist die Vererbung, bei der eine abgeleitete Klasse die Eigenschaften einer übergeordneten Klasse übernehmen soll. Jedoch existiert hier in VB auch ein Mechanismus, der über eine *Implements* Anweisung die Übernahme von Eigenschaften und Methoden einer Klasse ermöglicht, siehe hierzu auch[1].

Nachteile von VB sind die höhere Ebene, auf der in VB programmiert wird. VB wird hierdurch zwar einfacher zu bedienen, doch wird dies mit einer geringeren Geschwindigkeit des erstellten Programms bezahlt, obwohl in der neuesten, 6. Version der Sprache, der gleiche Codegenerator bzw. Microsofts neuester C++ Compiler zum Einsatz kommt. Als weiteren Nachteil ließe sich noch die Systemnähe von VB zu MS-Windows zu nennen. Es bietet sich keine Möglichkeit plattformunabhängig zu agieren[1].

Wie oben bereits erläutert, bestehen unterschiedliche Möglichkeiten der Anbindung der Funktionen des Softwarebaukastens an die verschiedenen Applikationen. Für die Office Anwendungen von Microsoft[17] auf der rechten Seite von Bild 1 kamen zwei Möglichkeiten der Anbindung in Frage, die aufgrund der unterschiedlichen Anforderungen auch beide realisiert wurden. Dies wird im Einzelnen in den nächsten Kapiteln näher erläutert.

### 1.4.1 Visual Basic for Applications[18] Macros

Ursprünglich war VBA die Bezeichnung der Makrosprachen für die Officeanwendungen. Mittlerweile wurden VB und VBA jedoch soweit angeglichen, dass auch die Kernfunktionen[19] von VB auf der VBA Bibliothek basieren. VB und VBA haben nahezu dieselbe Syntax, sowie dieselben Basiskommandos. Die maßgeblichen Unterschiede liegen darin, dass VBA keine eigene Programmiersprache ist und an die unterschiedlichen Anwendungen gebunden ist. Zudem besitzt VBA keinen echten Compiler und wird dadurch merklich langsamer. Weiterhin unterstützt VBA nicht alle Features die von VB unterstützt werden, was beispielsweise die Programmierung eigener Steuerelemente betrifft. Auch sind die in VBA verfügbaren Steuerelemente, die hier aus der MS-Forms-Bibliothek stammen und in einem eigenen Formular-Editor bearbeitet werden trotz äußerlicher Identität, nicht dieselben, die für gewöhnlich in VB Verwendung finden. Hier bestehen leider viele Inkompatibilitäten.

---

[14] ADO - ActiveX Data Objects. Neue Datenbank Objektbibliothek der Fa. Microsoft
[15] DCOM - Distributed Component Object Model
[16] Auch ‚late binding' genannt. Zur Zeit des Compilierens ist noch nicht bekannt, welches Objekt einer Variablen zugewiesen wird
[17] Innerhalb des Projekt fand das Packet Office 2000 Anwendung.
[18] Im Folgenden auch VBA genannt
[19] Unter Kernfunktionen sind hier die Komandos zur Variablenverwaltung gemeint.

Was die VBA-Programmierung erschwert, sind die unterschiedlichen Objekthierarchien der verschiedenen Anwendungen. Excel allein stellt ca. 100 verschiedene Objekte, die gemeinsam um die 1000 verschiedenen Eigenschaften und Methoden haben, was bei anderen Office Anwendungen nicht besser aussieht. [1] Sich in die Objekthierarchie der einzelnen Anwendungen einzuarbeiten bedeutet demnach keinen unerheblichen Aufwand.

Von Bild 1 ausgehend und vorgehend bereits erwähnt, wurde ein Teil der Funktionen des Softwarebaukastens direkt über VBA Macros an die Office Anwendungen angebunden.

Excel beispielsweise bietet die Möglichkeit über eine Funktionsdeklarierung ein Standard-eingabefenster zu generieren. Diese Möglichkeit wurde hauptsächlich bei Funktionen genutzt, welche eine überschaubare Anzahl an Eingabeparameter haben. Bei MS Access ist das Ganze etwas anders gestaltet und wird durch eigens zugeschnittene Formularoberflächen zwar benutzerfreundlich, jedoch auch nicht mehr so transparent, die sich dahinter verbergende Funktionalität ist jedoch dieselbe. MS Access bietet hier kein Standardeingabefenster, jedoch bietet Access eine relativ einfache Möglichkeit der Erstellung eigener Formulare.

Die Programmierung der Macros findet in einer eigenen Entwicklungsumgebung statt, die an die jeweilige Office Anwendung gebunden ist und auch aus dieser gestartet wird. Sie setzt sich aus dem Projekt-Explorer, dem Eigenschaftsfenster und dem Codeeditor zusammen. Die Entwicklungsumgebungen der verschiedenen Office Anwendungen sind unterscheiden sich nur unwesentlich. Bild 25 zeigt beispielsweise die Entwicklungsumgebung von VBA für Excel.

Im Projekt-Explorer sind alle VB Bestandteile einer Applikation angeordnet, wie zum Beispiel die Microsoft Excel Objekte, Module oder auch Klassenmodule. Über den Projekt-Explorer kann auch zwischen den einzelnen Bestandteilen der Application gewechselt werden. Das Eigenschaftsfenster zeigt alle Eigenschaften eines Objekts alphabetisch oder in Kategorien getrennt. Diese können hier auch gesetzt oder abgeändert werden. Im Codeeditor wird der Programmcode eingegeben und auch gleich auf syntaktische Richtigkeit geprüft.

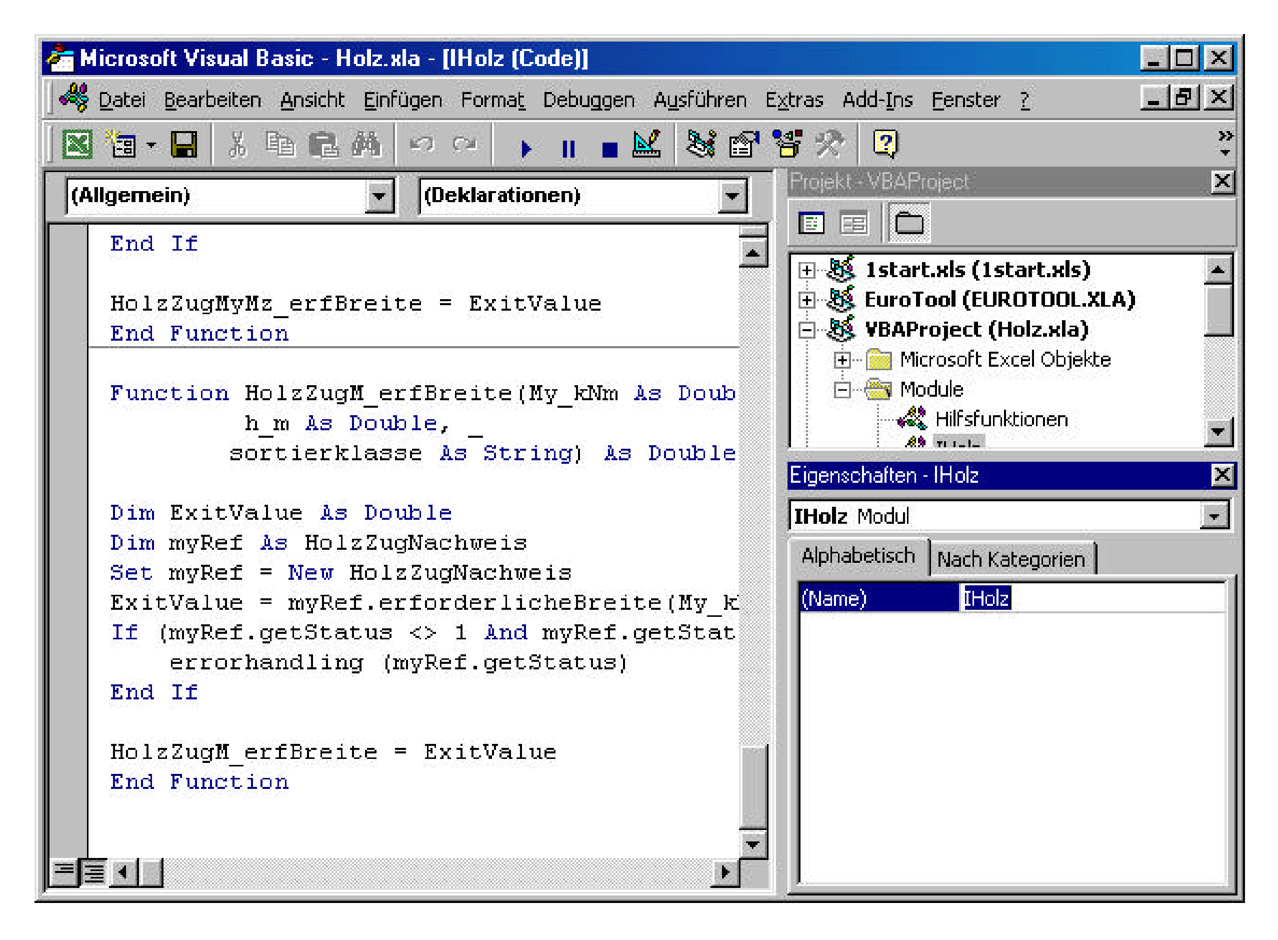



Bild 25. Entwicklungsumgebung von VBA in Excel

### 1.4.1.1 Verwendung von VBA Macros in Excel und das Formulardesign

VBA Makros sind die standardmäßige Möglichkeit für die Bereitstellung der Funktionen des Softwarebaukastens in Excel.

In Excel selbst wird auf die Funktionen über den Menübefehl *Einfügen/Funktion* zugegriffen. Hierbei wird ein Dialogfenster in Bild 26 geöffnet. Die erstellten Funktionen sind unter der Rubrik „Benutzerdefiniert“ zu finden.

Nach der Bestätigung wird ein Dialogfenster geöffnet (siehe Bild 27), das auf die programmierte Funktion zugeschnitten ist. Dies wird von Excel automatisch generiert, sobald ein VBA Macro existiert, das eine entsprechende Funktionsdeklaration besitzt, wie folgender Programmcode zeigt.

```
Function HolzZugM_erfBreite(My_kNm As Double, N_kN As Double, _
            h_m As Double, sortierklasse As String) As Double
```

In diesem Fall werden vier Eingabewerte benötigt, die in Klammer nach dem Funktionsaufruf stehen. Man spricht hier auch von Übergabeparametern, welche die Funktion erwartet.

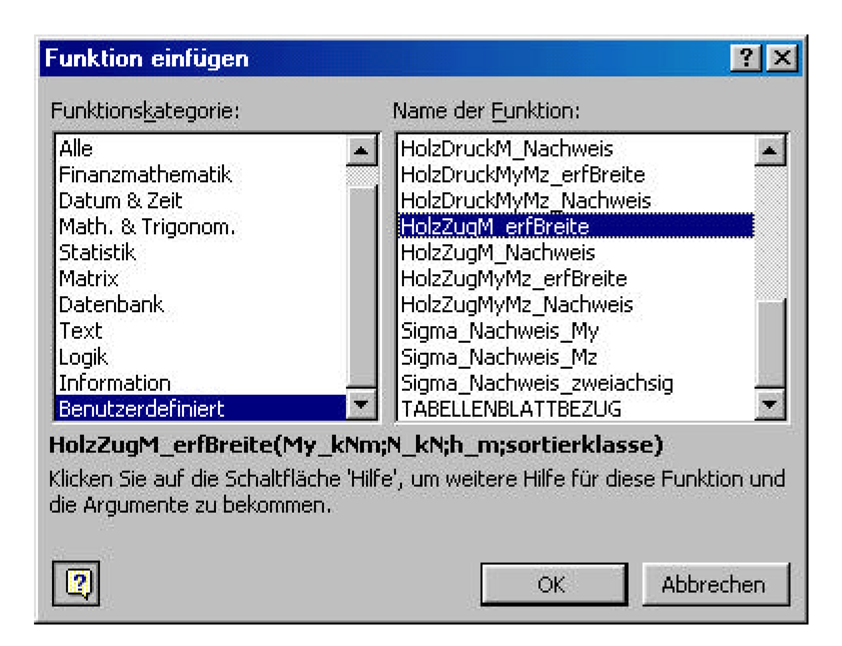



Bild 26. Dialogfenster *Einfügen Funktion* von MS-Excel

Nach der Bestätigung mit *OK* wird das Standarteingabefenster in den Speicher geladen, wie Bild 27 zeigt.

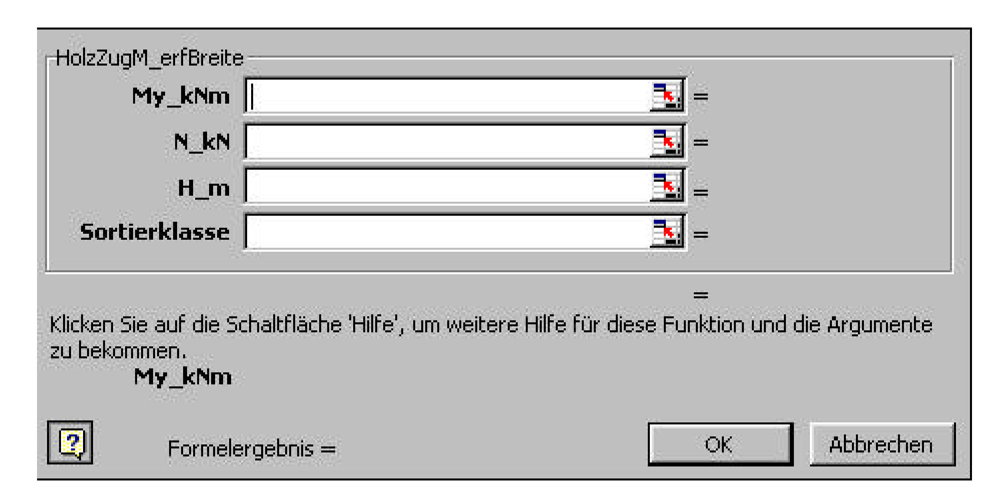



Bild 27. Von Excel erzeugtes Fenster für die Eingabe von Werten

Bei der Ausgabe in die markierte Zelle im aktuellen Excel Arbeitsblatt, ist zu beachten, dass hier lediglich ein Wert direkt angezeigt werden kann. Sind mehrere Ergebniswerte als Rückgabewerte vorhanden, beispielsweise in einem Array, muss der Anwender dann den gewünschten Wert über die *Index* Funktion von Excel anzeigen lassen. Sollen mehrere berechnete Werte auf einmal ausgegeben und angezeigt werden, muss dies entweder dann als String[20] geschehen, oder mehrere Funktionsaufrufe programmiert werden.

Weiterhin werden die Textfelder für die Eingabewerte immer in einer Liste angezeigt, siehe Bild 27. Die Eingabewerte können bei einer Vielzahl von Übergabewerten also nicht gruppiert angezeigt werden, was bei vielen Werten ein Problem für die Übersichtlichkeit bedeuten kann.

[20] Ein String ist eine Zeichenfolge. Datentyp in verschiedenen Programmiersprachen.

Zudem sind die Beschriftungsmöglichkeiten im Standarteingabefenster beschränkt und wie in Bild 27 im Fall des Übergabeparameters *Sortierklasse* ersichtlich ist, muss dem Benutzer ein Hinweis gegeben werden, in welcher Form oder Einheit die Werte eingegeben werden müssen. Deswegen wurde eine eigene Onlinehilfe benötigt, was unten näher erklärt wird.

Eine weitere Beschränkung ist die Anzahl der Übergabeparameter, diese kann nicht dynamisch, also während des Programmablaufs veränderlich erfolgen. Dies bedeutet, dass die Anzahl der Parameter im voraus bekannt sein muss, was jedoch nicht immer der Fall ist. Für die Eingabe eines Durchlaufträgers, der beispielsweise aus 2 aber auch aus 10 Feldern bestehen kann, wie es mit DLT[21] geschah, sind eine Vielzahl von veränderlichen Werten nötig. In solchen Fällen wurde auf eine andere Lösung gewählt, die Programmierung von Dll's als In-Process-Server.

Die einzelnen Funktionen werden in einem oder mehren Modulen in eine Excel Datei geschrieben und als Excel-Addin, mit der Dateierweiterung .xla abgespeichert. Die Speicherung erfolgt im Xlstart Verzeichnis der Excel Anwendung. Damit wird das AddIn beim Start von MS-Excel automatisch geladen und die Funktionen stehen zur Verfügung.

Die Vorteile bei der Speicherung als AddIn liegen in der Bereitstellung der Funktionen für andere Dateien. Während in herkömmlichen Excel Dateien, mit der Dateikennung .xls am Ende ein Verweis auf die Funktionen erstellt oder der Macroname vorangestellt werden muss, ist dies bei AddIns nicht mehr nötig. Weiterhin sind die Tabellenblätter eines AddIn in der Anwendung nicht mehr sichtbar. Außerdem bestünde die Möglichkeit den Code weitgehend gegen Eingriffe vom Benutzer zu schützen, indem man das Makro für die Bearbeitung mit einem Passwort schützt.

Hierin liegt allgemein gesehen ein großer Nachteil von Macros. Der Code von geladenen AddIns oder Macros ist mit der Entwicklungsumgebung der Anwendung einzusehen und kann bearbeitet werden und bei ungenügender Kenntnis auch unbrauchbar gemacht werden, was auch nicht im Interesse des Herstellers der Software ist. Es besteht demnach ein hinreichender Grund den erstellten Code zu schützen, was bei Macros und AddIns auch mit Schutz nur bedingt gegeben ist.

Wie oben bereits beschrieben, war bei Funktionen des Softwarebaukastens, welche eine dynamisch veränderliche Anzahl an Übergabeparametern benötigen, wie beispielsweise ein Durchlaufträger, die standardisierten Eingabemöglichkeiten von Excel nicht ausreichend. Benötigt wurde eine Eingabemöglichkeit für Variablen und Werte, die sich dynamisch verändern ließ und welche die Werte möglichst getrennt in Gruppen anzeigt, was die Übersichtlichkeit erhöht und am besten in einem Formular realisiert wird. Da in MS-Access eine relativ einfache und benutzerfreundliche Möglichkeit besteht Formulare zu entwerfen, wie unten jedoch noch erläutert wird, ergab sich dieser Umstand lediglich für die MS-Excel Anwendung.

Hierfür stehen mehrere Möglichkeiten zur Verfügung. Einerseits besteht die Möglichkeit ein Eingabeformular in einer Anwendungsdatei selbst zu generieren, welches als herkömmliche Datei mit der Erweiterung .xls gespeichert wird, dies wurde für verschiedene Holzbaunachweise getan oder wie Bild 28 zeigt, im Fall der Eingabe für die Berechnung von

[21] DLT – Durchlaufträger. Programmbibliothek mit Funktionen zur Berechnung der Schnittgrößen von beliebigen Durchlaufträgern.




zweidimensionalen Fachwerken. Innerhalb des Tabellenblatts können verschiedene Steuerelemente platziert werden, wie in Bild 28 mit den rot unterlegten Command-Buttons[22] zu sehen ist, denen mit VBA Code in Macros Funktionalität hinterlegt wird.

Unter Windows gibt es einige vordefinierte Funktionen oder auch Standarddialoge, die immer wieder benötigt werden. Die beiden einfachsten Standarddialoge in VB sind die Input-Box im unteren Bild und die MsgBox[23]. In einer Input-Box werden vom Benutzer Eingaben abgefragt, die über Programmcode ausgewertet werden. Eine MsgBox dient zum Anzeigen von Meldungen und Hinweisen für den Anwender. Dies sind keine fest integrierten Steuerelemente im Tabellenblatt selbst sondern werden aus dem Code während des Programmablaufs gestartet und erst dann in den Speicher geladen.

Weiterhin können in den Zellen des Tabellenblattes selbst ebenfalls Eingaben gemacht werden. Eine derart gestaltete Oberfläche ist durchaus geeignet, die oben genannten Aufgaben zu erfüllen und kann auch benutzerfreundlich gestaltet werden.

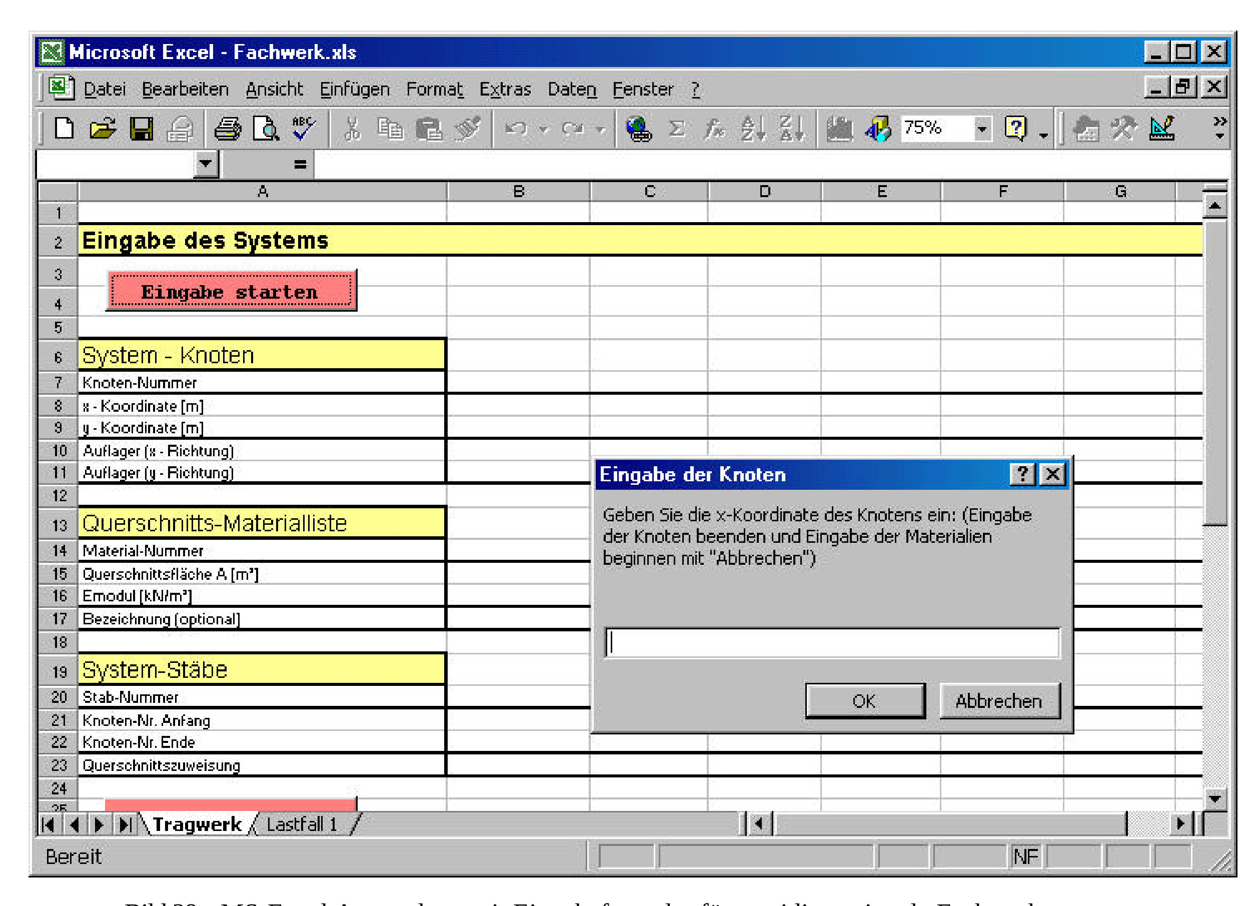



Bild 28. MS-Excel Anwendung mit Eingabeformular für zweidimensionale Fachwerke

Dennoch sind bei dieser Möglichkeit der Formularerstellung auch Grenzen gesetzt, was beispielsweise die Größe der Datei anbelangt und damit die Ladegeschwindigkeit beeinflusst. Es ist demnach nicht unbedingt sinnvoll, diese Art von Anwendung im Startverzeichnis von MS-Excel abzuspeichern, damit die Funktionalität sofort nach Programmstart genutzt werden kann. Es handelt sich hierbei auch um keinen einfachen Funktionsaufruf. Die Datei wird beim Öffnen in den Speicher geladen und ist somit nach dem Programmstart auch sichtbar.

[22] Command-Buttons – Befehlsschaltfläche, in Bild 28 aus der Microsoft-Forms-Bibliothek
[23] MsgBox – Message-Box Funktionsaufruf in VB. Stellt eine Möglichkeit

Ein weiterer Punkt, der in Betracht gezogen werden muss, ist die Geschwindigkeit, mit der VBA Code ausgeführt wird. Da VBA keinen echten Compiler besitzt, also kein Binärcode erzeugt wird, ist der Programmablauf vergleichsweise langsam. Bei großen Anwendungen wirkt sich dies unter Umständen nachteilig aus.

#### 1.4.1.2 VBA Macros in MS-Access und Formular design

Zur Verwendung kam die neueste Version der Anwendung, MS-Access 2000. Mit MS-Access hat die Firma Microsoft ein benutzerfreundliches Programm für relationales Datenbankmanagement auf den Markt gebracht. Die fensterorientierte Oberfläche erleichtert den Einstieg und die Arbeit mit Access erheblich. Bevor jedoch mit der Erstellung einer Datenbank begonnen wird, sollten vorab Überlegungen über die Architektur der Datenbank stattfinden, was jedoch nicht Thema dieser Arbeit ist und wo auf weiterführende Literatur verwiesen wird, wie beispielsweise [8].

Die Programmierung der VBA Macros findet, wie in MS Excel, in einer eigenen Entwicklungsumgebung statt, die an Access gebunden ist und die auch aus dieser gestartet wird. Sie setzt sich wie in Bild 25 für die Umgebung in Excel, aus dem Projekt-Explorer, dem Eigenschaftsfenster sowie dem Codeeditor zusammen.

Beim Zugriff auf die Dll's des Softwarebaukastens, beziehungsweise bei der Initialisierung von Objekten über die COM-Technologie ergeben sich grundsätzlich keine Unterschiede zu Excel. Es wird eine Objektvariable deklariert, die später eine Referenz auf eine die Java programmierte COM-Klasse oder COM-Objekt erhält. Hierzu wird wieder das Schlüsselwort *Set* in Verbindung mit *new* benötigt. So wird an dieser Stelle auf ein Codebeispiel verzichtet.

MS Access bietet, im Gegensatz zu MS Excel, keine standardmäßig erzeugten Fenster, in denen Eingaben gemacht werden können. Jedoch bietet Access eine relativ einfache Möglichkeit eigene Benutzeroberflächen zu erzeugen. Dabei steht ein eigener Assistent zur Verfügung. Weiterhin kann eine Vielzahl an Steuerelementen eingefügt werden, die aus den unterschiedlichsten Bibliotheken stammen können. Für Formulare ist das die MS-Form-Bibliothek, wofür MS Access einen eigenen Formulareditor besitzt, wie Bild 29 im Fall von Berechnungsmöglichkeiten für den Holzbau zeigt.

Die Verwendung von selbst erstellten Formularen erleichtert das Arbeiten in MS Access sehr. Die Formulare sind nichts anderes als die individuelle Form der Tabellen- und Abfragedarstellung für die Datenbank. Die Daten werden in eine grafische Oberfläche eingebunden, was die Darstellung übersichtlicher und das Arbeiten komfortabler macht.

Die Formulare können nicht nur Tabellen- und Abfragewerte haben, sondern es können, wie oben bereits erwähnt, eine Vielzahl verschiedener Steuerelemente zur Anwendung kommen. So sind beispielsweise Schaltflächen aus der MS-Form-Bibliothek zur Ausführung von Macros oder Funktionen möglich. Der Zweck eines Formulars kann somit sowohl die Ein- und Ausgabe von Werten als auch das Auslösen von Befehlen sein [9].

Wie eingangs bereits erwähnt haben die unterschiedlichen Office-Anwendungen ihre eigene Objekthierarchie, auf die jedoch hier im Näheren nicht eingegangen wird. Hierzu sei beispielsweise auf [10] verwiesen.

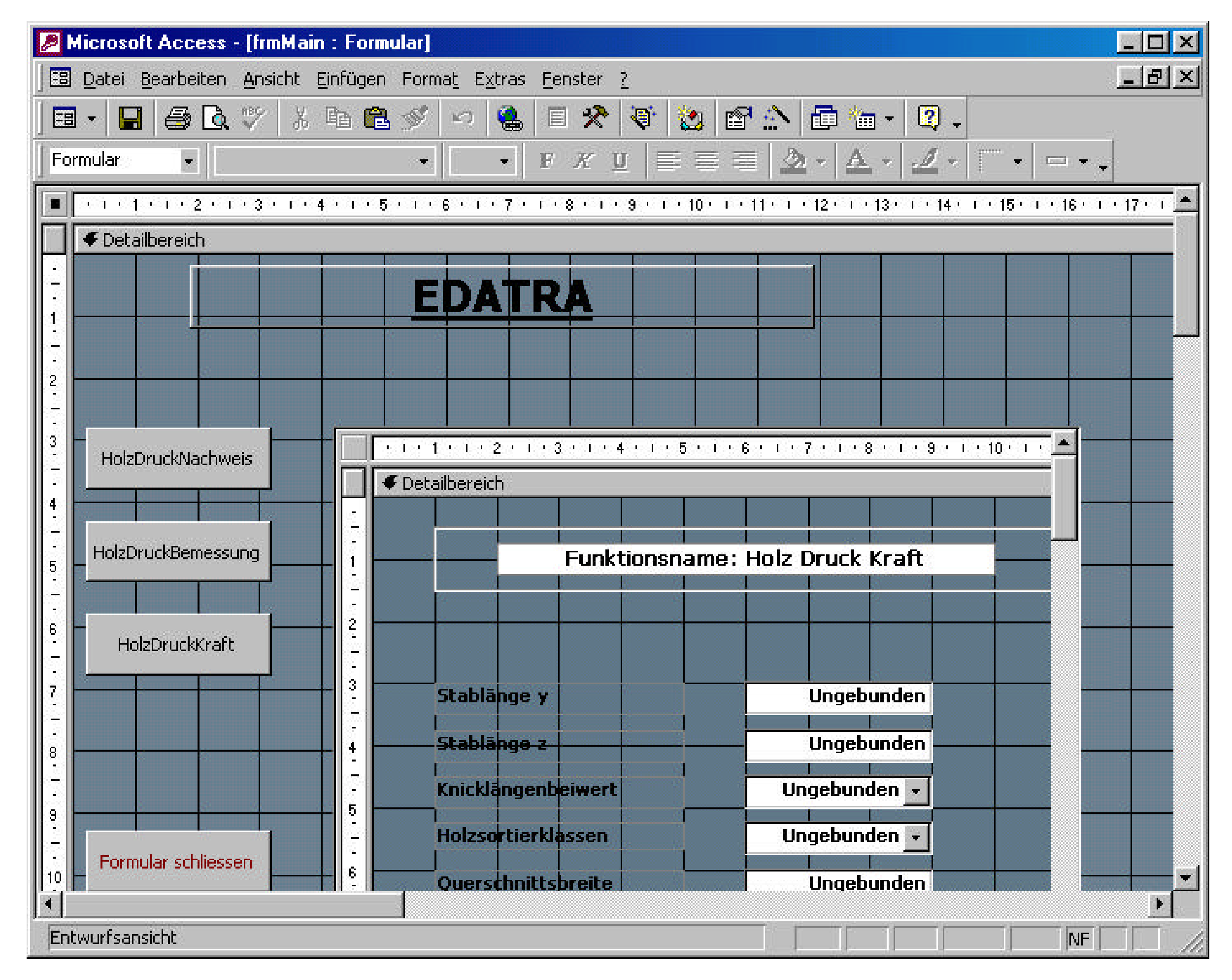



Bild 29. MS Access in der Entwurfsansicht für ein Formular

Um auf einzelne Eigenschaften zuzugreifen, um sie zu bearbeiten, kann entweder über die rechte Maustaste ein Dialogfenster geöffnet werden, mit dem auf die einzelnen Eigenschaften der Steuerelemente zugegriffen wird, wie Bild 30 zeigt oder die Eigenschaften können direkt in der VBA Entwicklungsumgebung im Eigenschaftsfenster bearbeitet werden.

MS Access bietet hier für einige Steuerelemente, wie beispielsweise Auswahllisten, eine Eigenschaft mit an, in der dem Steuerelement ein Datensatz hinterlegt werden kann. MS-Access wurde also bereits sehr benutzerfreundlich gestaltet. Nach einer Einarbeitung in die Objekthierarchie ist es möglich in relativ kurzer Zeit eine benutzerfreundliche Oberfläche zu gestalten, welche auf die speziellen Anforderungen der jeweiligen Datensätze abgestimmt ist. Hierin liegt beispielsweise gegenüber MS-EXCEL ein großer Vorteil, insbesondere, wenn die Anzahl der Eingabewerte variabel ist.

Befehlsschaltfläche: btHolzDruckNachweis

Format | Daten | Ereignis | Andere | Alle

| | |
|---|---|
| Beschriftung | HolzDruckNachweis |
| Bild | (keines) |
| Bildtyp | Eingebettet |
| Transparent | Nein |
| Hyperlink-Adresse | |
| Hyperlink-Unteradresse | |
| Sichtbar | Ja |
| Anzeigen | Immer |
| Links | 1cm |
| Oben | 5cm |
| Breite | 3cm |
| Höhe | 1cm |
| Textfarbe | -2147483630 |

Bild 30. Dialogfenster für Eigenschaften aus MS Access

### 1.4.2 Visual Basic für Serveranwendungen

#### 1.4.2.1 Allgemeines

Aufgrund der in den vorangegangenen Kapiteln erläuterten Notwendigkeit für eine ansprechende Oberfläche für bestimmte Softwaremodule und der Restriktionen, die sich für herkömmliche Dateien in MS-Excel ergaben, wurde eine Möglichkeit gesucht, diese Funktionalität, möglichst einfach, auf andere Art bereit zu stellen. In Kapitel 1.1 und Kapitel 1.2 wurden verschiedene Eigenschaften der Serveranwendungen erläutert. Weiterhin sollten die Funktionen in Java Code geschrieben sein. Benötigt wurde also lediglich eine Oberfläche, in die Daten eingegeben, kontrolliert und verändert werden können.

Um Oberflächen und Formulare zu erstellen bietet sich Visual Basic durch die visuelle Programmierung, bei der per Drag and Drop[24] Steuerelemente auf Formularen platziert und verändert werden können, an. Für die Erstellung der Anwendung gibt es in VB hier wieder mehrere Möglichkeiten. Da vorab die Bereitstellung der Oberfläche nur für MS-Excel erfolgen sollte, wurde die Anwendung als Dll, bzw. als In-Process-Server erstellt. Im Grunde genommen sind Dll's jedoch nichts anderes als Windows Programme. Die wesentlichen Unterschiede liegen darin, dass eine Dll nicht für sich gestartet werden kann und über ein anderes Programm, das die Funktionen der Dll, bzw. des Servers, nutzen möchte, aufgerufen werden muss.

Eine weitere Möglichkeit besteht darin, die Oberfläche als Out-of-Process-Server, was auch als ActiveX-Exe bezeichnet wird, zu programmieren. Der Server hat hier den Charakter einer selbständigen .exe Datei, d.h. die Anwendung wird automatisch gestartet, wenn ein Programm die Funktionen des Servers nutzen will. Der Server läuft dann in einem eigenen Thread und hat einen eigenen Adressraum. Dies hat den Vorteil, dass vom Server Aufgaben erledigt werden können, ohne, dass der Client auf dessen Abschluss warten muss. Dies wird auch als Multithreading bezeichnet. Bei den hier implementierten Funktionen waren jedoch Probleme mit der Rechenzeit nicht zu erwarten, da ihr Rechenumfang gering ist und sie in kompilierter Form als Binärcode aufgerufen werden.

[24] Drag and Drop bedeutet das Aufnehmen und Ziehen eines Steuerelements mit der Maus und Platzieren an der gewünschten Stelle

Sinnvoll wäre hingegen die Kompilierung zu einer ActiveX-Exe, wenn beispielsweise die erstellte Oberfläche für mehrere Office-Anwendungen zur Verfügung stehen soll. Man könnte dann die Anwendung, welche die Funktionalität der Oberfläche bereit stellt, aus einer Office-Anwendung heraus starten, zum Beispiel aus MS-Excel, verschiedene Werte übernehmen, MS-Excel schließen und die Ergebnisse nach der Berechnung in einer anderen Anwendung entsprechend formatiert ausgeben. Die Anwendungsmöglichkeiten sind also vielfältig.

Da man auch für diese Art der Nutzung offen bleiben wollte, wurden die Komponenten, welche die Oberflächen bereitstellen, zwar für die bisher verlangten Features sinnvollerweise als Dll, also In-Prozess-Server kompiliert, der interne Aufbau wurde jedoch so gewählt, dass auch ohne größere Veränderungen im Code ein Out-of-Process-Server daraus erstellt werden könnte. Da auf den Code der jeweiligen Komponenten an entsprechender Stelle noch eingegangen wird, wird an dieser Stelle lediglich die Erstellung aufgezeigt, bzw. die entsprechenden Compilereinstellungen erläutert.

### 1.4.2.2 Erstellung einer Dll mit Visual Basic

Die Entwicklungsumgebung von Visual Basic ist grundsätzlich wie die bisher vorgestellten Entwicklungsumgebungen aufgebaut.

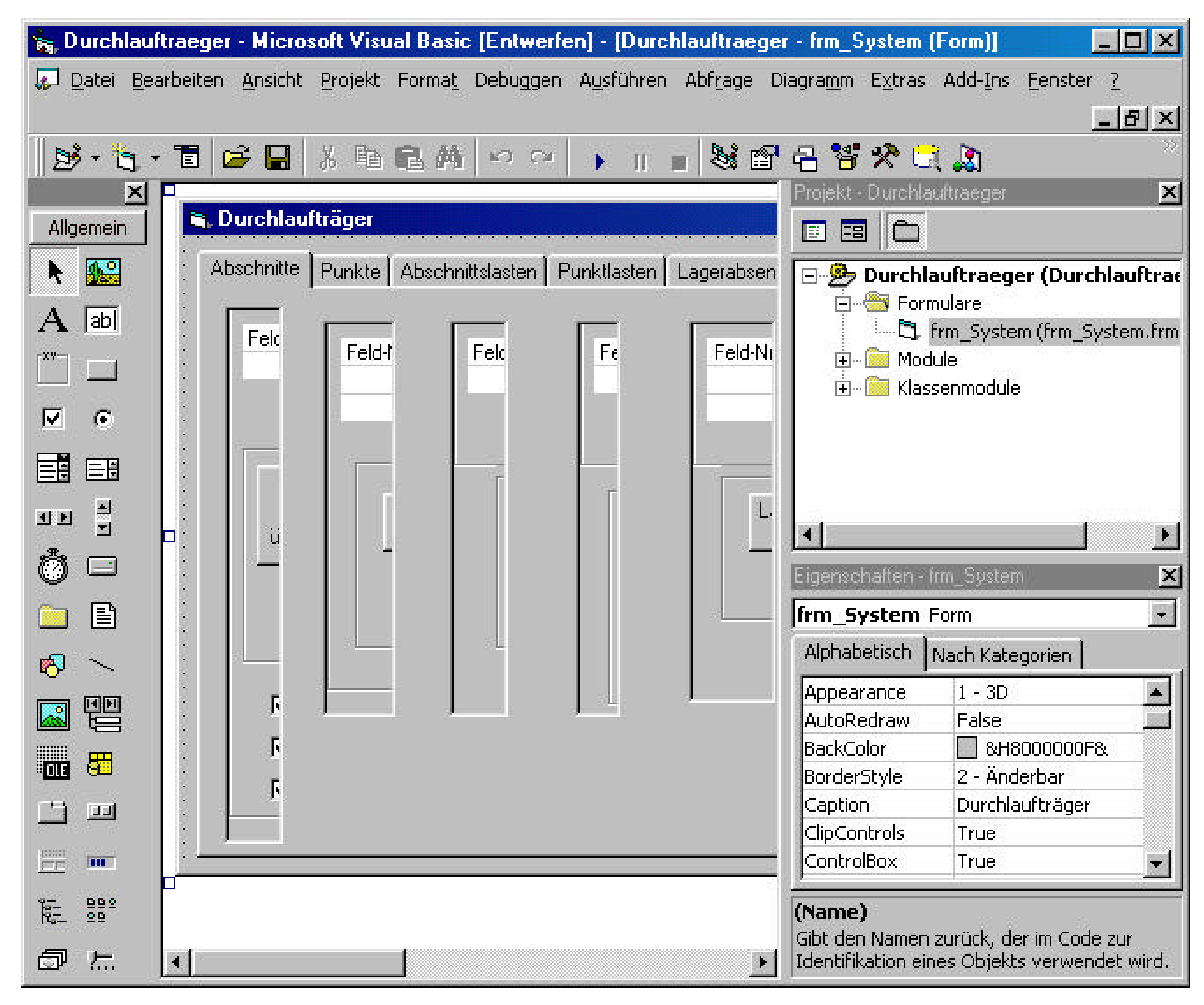



Bild 31. Die Entwicklungsumgebung von Visual Basic

Sie setzt sich hauptsächlich wieder aus dem Projekt-Explorer, dem Eigenschaftsfenster und dem Codeeditor zusammen, was Bild 31 zeigt. Durch den mehr oder minder gleichen Aufbau und dasselbe Layout der Entwicklungsumgebungen der Fa. Microsoft in den verschiedenen Produkten, wird deutlich, dass sich ein Programmierer sehr schnell in einer neuen Umgebung zurecht findet, wenn er sich bereits mit einer auskennt. Es können auch noch weitere Elemente wie Symbolleisten in die Entwicklungsumgebung aufgenommen werden wie Bild 31 mit der Tool-Box auf der linken Seite zeigt.

Nach dem Programmstart von Visual Basic erscheint ein Eröffnungsdialog (Bild 32), in dem bei Erstellen eines neues Projekts die Projekteigenschaften eingestellt werden können. Diese können jedoch auch noch nachträglich über das Dialogfenster Projekteigenschaften verändert werden.

Wenn eine ActiveX-Dll erstellt wird, müssen auch noch weitere Dinge beachtet werden, wie beispielsweise das Threading-Model. Das Listenfeld Threading-Modell (Bild 33) wird nur bei ActiveX.dlls und ActiveX-Steuerelementprojekten aktiviert. Die einzigen Optionen sind Single-Threaded und Apartment-Threaded.

Zwar unterstützen VB Dll's von sich aus kein Multithreading, dennoch ist die Einstellung zu beachten, für den Fall, dass die Dll von einem Multithreading-Prozess genutzt wird. Single-Threaded bedeuted, dass alle Aufrufe von Eigenschaften oder Methoden der Dll der Reihe nach ausgeführt werden, Apartment-Threaded zeigt an, dass globale Daten der Dll für jeden Thread neu angelegt werden. Die Methoden der Dll können dann von jedem Thread gleichzeitig ausgeführt werden [1].




Bild 32. Dialogfenster Neues Projekt der Entwicklungsumgebung von Visual Basic

Weiterhin muss die Dll wenigstens eine Klasse besitzen, deren Instancing Eigenschaft auf MultiUse gesetzt ist, siehe Bild 34. Die Instancing Eigenschaft ist dafür zuständig, wie die Klasse von außen benutzt werden kann.

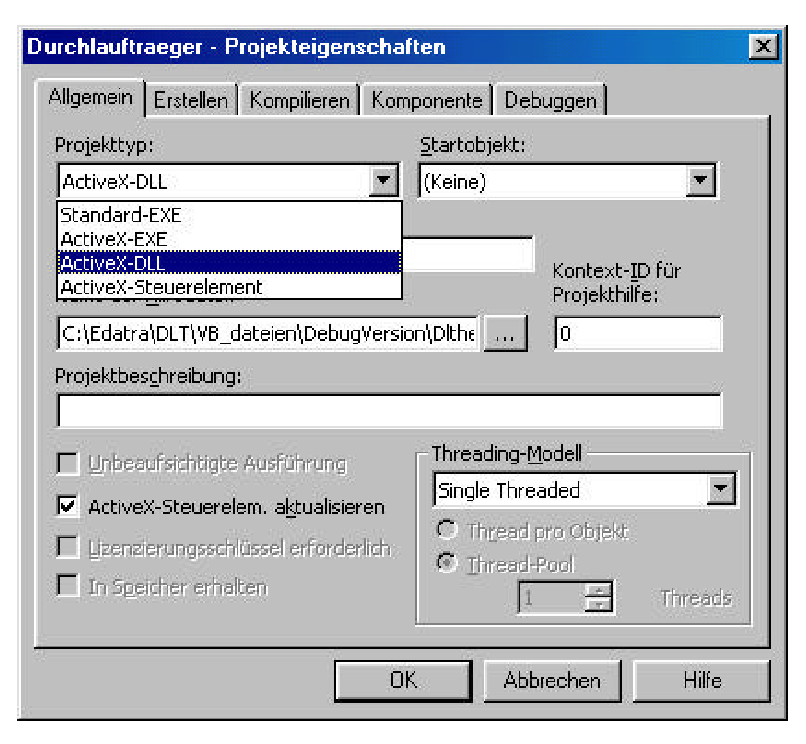



Bild 33. Projekteinstellungen in VB für ein erstelltes Projekt

MultiUse oder auch InSameProcessMultiUse ermöglicht anderen Anwendungen das Erstellen von Objekten aus der Klasse. Eine Instanz Ihrer Komponente kann eine beliebige Anzahl von auf diese Weise erstellten Objekten bereitstellen. Eine weitere Einstellmöglichkeit wäre auch GlobalMultiUse, das wie MultiUse funktioniert, jedoch mit dem Unterschied, dass die Eigenschaften und Methoden der Klasse aufgerufen werden können, als ob es sich um einfache globale Funktionen handeln würde. Es ist nicht erforderlich, dass zuerst explizit eine Instanz der Klasse erstellt wird, da eine Instanz automatisch erstellt wird [5].

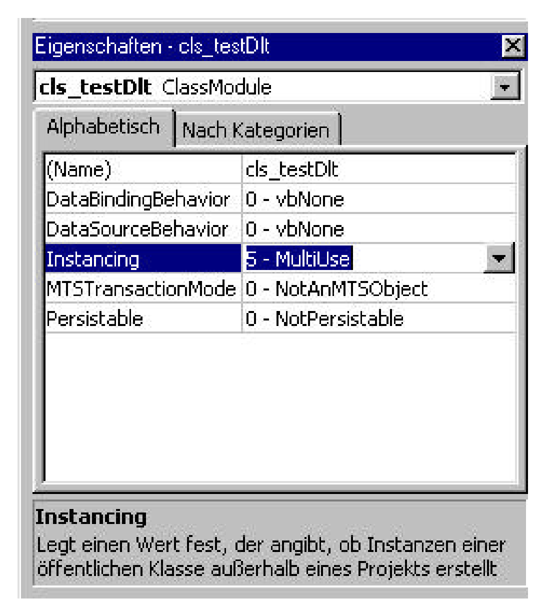



Bild 34. Die Instancing Eigenschaften einer Visual Basic Dll

Es existieren noch weitere Einstellungen, wie PublicNotCreatable, durch die andere Anwendungen Objekte dieser Klasse nur verwenden können, wenn die Komponente das Objekt zuerst erstellt. Andere Anwendungen können die CreateObject-Funktion oder den New-Operator also nicht verwenden, um Objekte aus der Klasse zu erstellen. Als letzte mögliche Einstellung für eine Dll in VB kann eine Klasse natürlich noch privat sein, was bedeutet, dass sie nicht instanziert werden kann. Diese beiden Möglichkeiten sind jedoch nicht relevant.

Die anderen Eigenschaften werden wie in Bild 34 als Defaultwerte eingestellt. Für weitere Information hierzu wird auf [5] oder [1] verwiesen.

## 1.5 Onlinehilfe für den Softwarebaukasten

In moderneren Programmen ist es selbstverständlich, dass der Anwender die momentan benötigte Hilfe zum geöffneten Fenster erhält, wenn die Taste *F1* oder ein Hilfe-Button betätigt wird. Der Anwender bekommt auf Anfrage Informationen zu dem Teil des Programms, mit dem er sich gerade befasst. Hierbei spricht man von kontextsensitiven Hilfe.

So sollte auch der Softwarebaukasten mit einer Onlinehilfe ausgestattet werden, die Informationen zu der entsprechenden Funktion liefert. Dies ist aufgrund der Menge an unterschiedlichen Funktionen, der Vielzahl von Parametern und vor allem den unterschiedlichen Einheiten der Ein- und Ausgabewerte erforderlich.

Die Anwendung Mathcad bietet hier wieder eine eigene Möglichkeit, wie der Hilfetext für eine Funktion zur Verfügung gestellt wird. Innerhalb einer XML-Datei, welche für die Anbindung ohnehin nötig ist, werden die Informationen bereit gestellt. Dieser erscheint dann wie Bild 44 zeigt und in Kapitel 1.6.2 nochmals näher erläutert wird.

Für die Office Anwendungen musste, vor allem wegen der ungenügenden Möglichkeiten, die Excel in Bezug auf die Bereitstellung von Hilfetext in den Standart Eingabefenstern ermöglicht, eine eigene Hilfe geschrieben werden. Hierzu gibt es mehrere Möglichkeiten. Für MS-Access ergab sich dieses Problem nicht, da ohnehin eine Oberfläche erstellt wurde, in der genügend Informationen bereit gestellt werden können.

Schließlich wurde die Onlinehilfe mit dem herkömmlichen Microsoft Help Workshop HCW generiert und kompiliert. Der Text wurde hierzu im RichText Format erstellt, in das Programm HCW geladen und weiter bearbeitet. Die Nutzung von HCW hat den Vorteil, dass die erstellte Hilfe auf jedem Windows Betriebssystem zur Verfügung steht. Das neuere HTML-Format einer Hilfedatei, das beispielsweise mit dem Programm HTMLHelp erstellt werden kann und durch die komfortableren Gestaltungsmöglichkeiten das ältere Rich Text Format mehr und mehr in den Hintergrund drängt, erfordert beim Benutzer die Installation des Microsoft Internet Explorers. Da jedoch auch heute noch nicht davon ausgegangen werden kann, dass dieser auf jedem Rechner installiert ist, wurde das herkömmliche RichText Format gewählt. Ein weiterer Vorteil von Hilfen, die mit HCW erstellt wurden ist, dass jedem Benutzer, der mit Microsoft Windows vertraut ist, die Hilfeumgebung hinreichend bekannt sein dürfte. Diese ist in allen Anwendungen gleich aufgebaut ist und beinhaltet auch eine Suchfunktion und einen Indexkatalog.

HCW und HTMLHelp sind Produkte der Firma Microsoft und kommen bei umfangreichen Hilfeprogrammen schnell an ihre Grenzen [1]. Jedoch existiert auch noch eine Vielzahl an

professionellen Werkzeugen von Drittanbietern, wie DocToHelp 2000 [11], Produkte der Firma Sinterphase [12] oder RoboHelp [13]. Diese Möglichkeiten wurden jedoch nicht in Betracht gezogen, da die Funktionalität von HCW für die Zwecke des Projekts völlig ausreichend war.

Das Hauptproblem bei der Erstellung einer Online Hilfe zu einer Applikation besteht darin, den Hilfetext zu schreiben, was einige Zeit und Mühe kostet, da der Text auch visuell mit anschaulichen Bildern aufgelockert werden sollte. So ist es nicht unüblich, dass Programmentwickler das Erstellen einer Online Hilfe zu einer Applikation an Fremdfirmen weitergeben oder keine Hilfe bereitstellen.

Um die Hilfedatei im herkömmlichen RichText Format zu erstellen werden zwei weitere Programme benötigt. Ein Textverarbeitungsprogramm, das in der Lage ist, Dateien im RichText Format zu speichern und der Hilfecompiler HCW, der die erstellte RichText Datei derart kompiliert, dass sie das Erscheinungsbild bekommt, wie es im Windows Betriebssystem üblich ist.

Der Hilfecompiler HCW wird mit der Visual Basic 6.0 CD-Rom ausgeliefert (<CD-Rom Laufwerk>:\Common\Tools\VB\HCW\). Das Textverarbeitungsprogramm wird als Editor benötigt, um den Hilfetext zu erstellen, da das Compilerprogramm HCW keinen eigenen Editor besitzt, was ein Manko ist. Als Editor wurde MS-Winword verwendet.
Eine Online Hilfe stellt demnach eine eigene Applikation dar, die mit einem Editor und einem speziellen Programm kompiliert werden muss. Dies bedeutet wiederum Fehlersuche und Überprüfung, ob ein reibungsloser Ablauf der vielschichtigen Verknüpfungen innerhalb der Hilfe gegeben ist. Sprungmarken müssen definiert werden, jedem Thema muss eine Kontextnummer zugewiesen sein, ein Inhaltsverzeichnis sollte vorliegen. Diese stellt wiederum eine eigenständige Datei dar, die nicht in die eigentliche Hilfedatei integriert ist. Dies muss auch bei der Erzeugung der Installationsdateien beachtet werden. Diese Umstände verdeutlichen, warum das Bereitstellen einer Hilfe bei Programmentwicklern ein leidiges Thema ist, das gerne umgangen wird. Die Erzeugung einer guten und benutzerfreundlichen Online Hilfe kostet sehr viel Zeit und Arbeitseinsatz. Nicht ohne Grund spezialisieren sich ganze Firmen nur auf das Erstellen von Hilfedateien.

### 1.5.1 Erzeugen des Hilfetextes

Bild 35 zeigt eine Seite der RichText Datei für die Hilfe zum Durchlaufträgerprogramm Dlt. Der Kontext für jede einzelne Funktion steht, durch einen Seitenwechsel getrennt, auf einer eigenen Seite. Die Fett gedruckte Überschrift wird durch vorangestellte Fußnoten ergänzt, welche vom Compiler HCW erkannt und interpretiert werden. Hierin bedeuten $ die Deklaration der Überschrift, # den Identifikationsnamen, dem später im Compiler eine Kontextnummer zugewiesen wird.. Im unteren Bildbereich sind die Fußnoten zu sehen.

Über ein vorangestelltes Fußnotenzeichen K kann zusätzlich noch definiert werden, welche Schlüsselwörter auf dem Tabellenblatt „Index“ der fertigen Hilfe in der Hilfedialogbox angezeigt werden sollen.

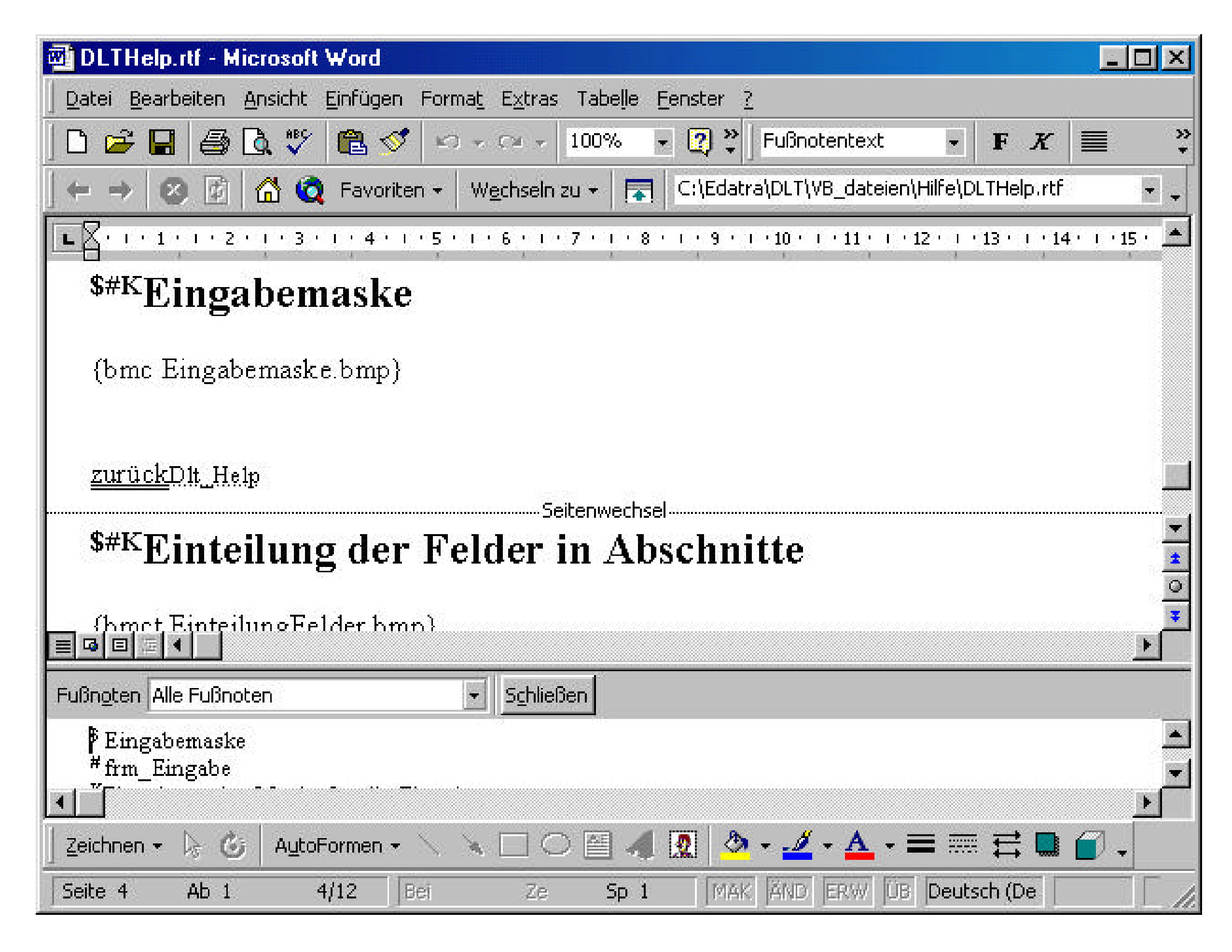



Bild 35. Textdatei für die Hilfe zum Durchlaufträgerprogramm

Im Text der Hilfedatei von Bild 35 ist die Deklaration einer Sprungmarke, sogenannten Hyperlinks zu sehen. Diese werden mit einer doppelten Unterstreichung und dem nachgestellten Schlüsselwort definiert. Doppelt unterstrichene Wörter werden später grün unterlegt angezeigt, wobei die nachgestellten Schlüsselwörter unsichtbar bleiben. Nach dem Klicken auf das unterlegte Wort wird an die angegebene Stelle in der Hilfedatei gesprungen.

Der Verweis auf Bilder, die in der Hilfe später angezeigt werden sollen, müssen in einer geschwungenen Klammer stehen. Mit dem Schlüsselwort *bmc* (Bitmap centered) und dem nachgestellten Namen der Bilddatei wird später im Compiler ein Verweis auf die Bilddatei erstellt und definiert, wie sie angezeigt werden soll.

Der Zugriff auf die kompilierte Hilfedatei erfolgt für die Standardeingabefenster in MS-Excel über die Einstellung ‚Name der Hilfedatei' im Dialogfenster Projekteigenschaften, wie Bild 36 für die Onlinehilfe für Funktionen des Stahl- und Holzbaus zeigt. Aufgerufen wird die Hilfe dann im Standartfenster für die Funktion über den Hilfebutton, der standardmäßig zum Formular hinzugefügt wird (Bild 27).

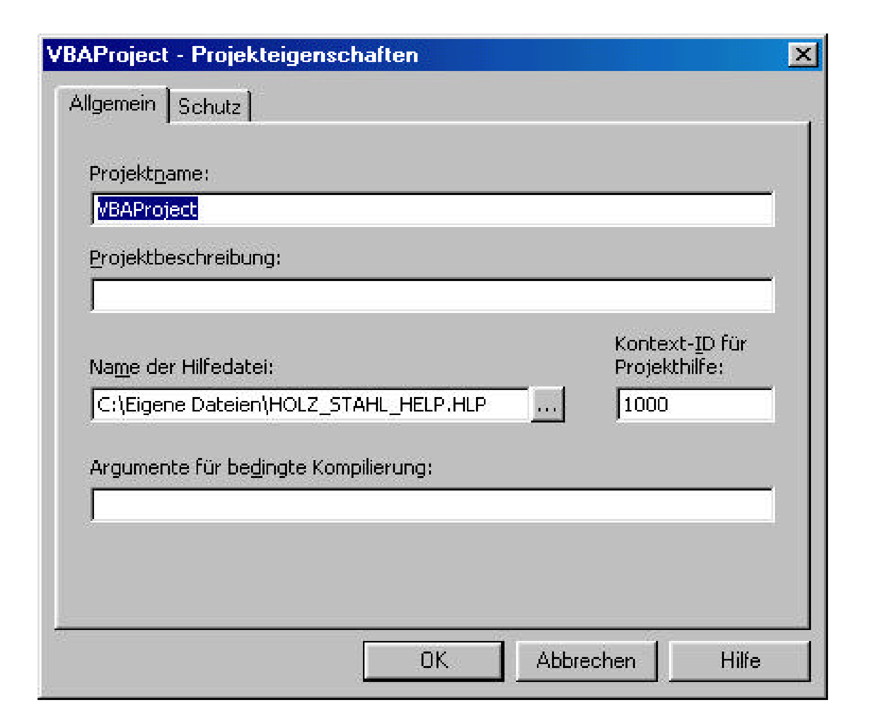



Bild 36. Zugriff auf eine Onlinehilfe für ein Standartfenster in MS-Excel

Für den Fall, dass eine eigene Oberfläche erstellt wurde, muss auf dem Fenster eine Befehlsschaltfläche vorhanden sein, der Code hinterlegt ist, über den auf die Hilfe zugegriffen wird. Dies war für das Programm zur Berechnung von Durchlaufträgern der Fall.

Um diese Möglichkeit der Einbindung einer Hilfedatei nutzen zu können wird außer der Befehlsschaltfläche noch ein CommonDialog[25] Steuerelement benötigt, über das die Windows-Hilfe angesprochen und aufgerufen werden kann. Folgender Programmcode zeigt das Einbinden der erstellten Hilfedatei in die Anwendung über das CommonDialog Steuerelement mit dem Namen comdial_dlt und den durch einen Punkt getrennten Eigenschaften des Objekts. Die Syntax des Zugriffs auf die einzelnen Eigenschaften ist also wieder dieselbe, wie allgemein bei Objekten, was auch hier wieder durch die COM-Technologie ermöglicht wird.

```
Zeile 1:   Private Sub cmd_help_Click()              'Startmit F1 oder den Hilfemenübefehl
Zeile 2:   comdial_dlt.HelpFile = "Dlthelp.hlp"      'Verweis auf die generierte Hilfedatei
zum
Zeile 3:   comdial_dlt.HelpContext = 1000            'Verweis auf das Eingangsfenster der Hilfe
Zeile 4:   comdial_dlt.HelpCommand = cdlHelpIndex    'Zeigt die Hilfe für den Kontext an
Zeile 5:   comdial_dlt.ShowHelp                      'Zeigt den MS Windows spezifischen Hilfe
Dialog an
Zeile 6:   End Sub
```

Programmlisting 3.

[25] Ein vom Betriebssystem Windows bereit gestelltes Standardsteuerelement das als Platzhalter für verschiedene Standarddialoge, wie *Speichern unter*,*Datei öffnen*, *Drucken* und dergleichen dient

## 1.5.2 Entwicklungsumgebung des Hilfecompilers

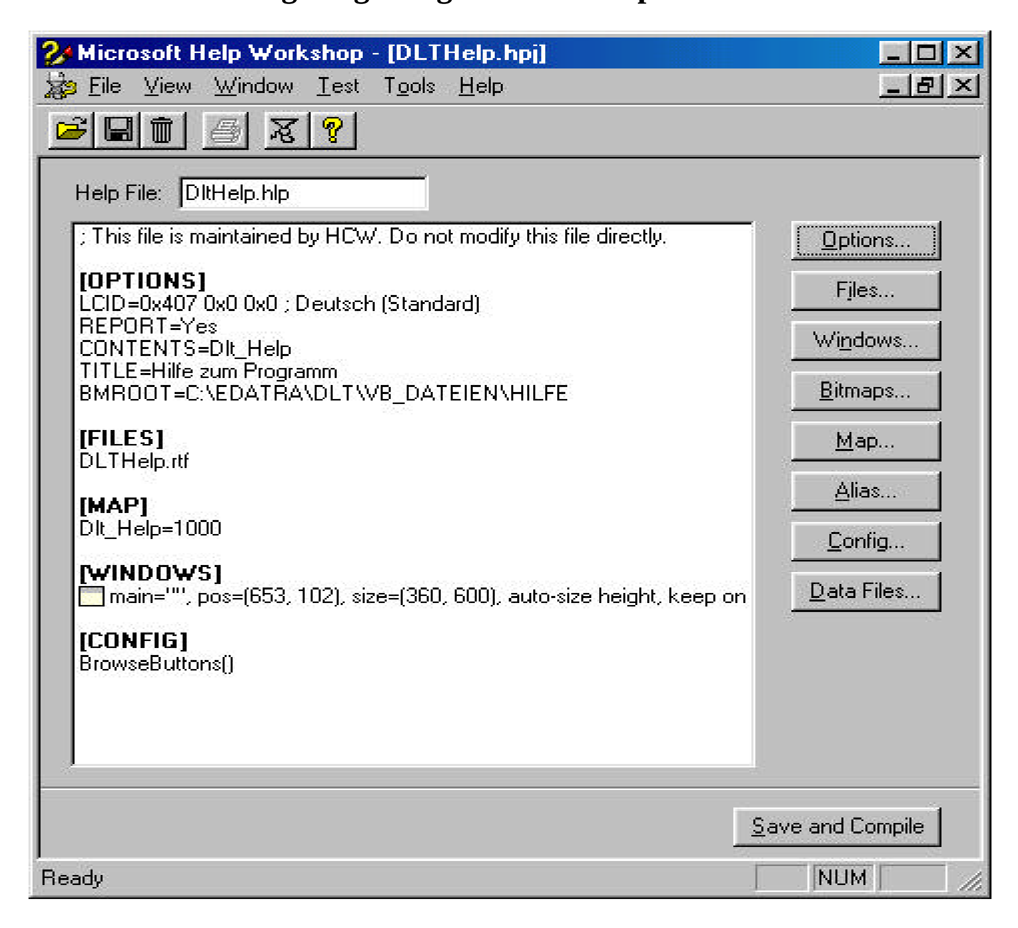



Bild 37. Die Entwicklungsumgebung von HCW

Nachdem der gesamte Text, den die Hilfe später erhalten soll, geschrieben wurde, kann in der Entwicklungsumgebung des Compilers *HCW* (siehe Bild 37) die weitere Bearbeitung vorgenommen werden. Er ist denkbar einfach aufgebaut, was auch für die verfügbare Hilfe gilt, falls genauere Informationen gesucht werden. Dies ist allerdings ein Nachteil. Weitere Nachteile sind auch der fehlende Editor, in dem der Text direkt in der Entwicklungsumgebung geschrieben werden kann sowie der Umstand, dass ein in der Entwicklungsumgebung erzeugtes Inhaltsverzeichnis der Hilfe nicht automatisch in die Hilfedatei integriert wird.

Hier werden allgemeine Dinge der Hilfedialogbox eingestellt, wie beispielsweise die Farbe, verfügbare Buttons und dergleichen. Im Bereich mit der Überschrift *Map* ist lediglich eine Zeile zu sehen, mit der dem Schlüsselnamen der ersten oder Startseite der RichText Hilfedatei die Kontextnummer 1000 zugewiesen wird. Für diese Seite, die im Programm über einen Button oder die Taste *F1* direkt angezeigt werden soll, muss auf diese Weise eine Kontextnummer vergeben werden.

Wenn alle Einstellungen und Zuweisungen erledigt wurden, kann die Hilfedatei generiert werden und nach der Fehlerkontrolle in die Anwendung eingebunden werden, was oben erklärt wurde.

## 1.6 C/C++

Die Anbindung der benutzerdefinierte Funktionen in Mathcad müssen mittels einer speziellen DLL in C beziehungsweise C++ realisiert werden. Bis auf die Installation der dazu notwendigen Dateien im richtigen Verzeichnis sind hierfür seitens des Anwenders keine Einstellungen mehr nötig. Mathcad erkennt beim nächsten Start automatisch alle neuen Funktionen und stellt sie in jedem Arbeitsblatt zur Verfügung. Der Aufruf erfolgt, wie bei jeder Standardfunktion, über den *Funktion einfügen* Dialog.

An dieser Stelle ist noch eine Besonderheit von Mathcad zu erwähnen. Der Hersteller dieser Software sieht es leider nicht vor, Zeichenketten (wie z.B. Holzartbezeichnungen oder Profilbezeichnungen für Stahlprofile) als Parameter einer Funktion anzugeben. Aus diesem Grund war es notwendig, eine andere Lösung für das Identifizieren der Materialarten und unterschiedlichen Querschnitte zu finden. In den realisierten Funktionen wurden dazu zwei Wege beschritten.

Einerseits wurde bei manchen Funktionen eine Indexnummer festgelegt, die beim Funktionsaufruf mit übergeben werden muss. Dieser Indexnummer ist ein spezielles Material zugewiesen und wird vor der Berechnung innerhalb der Dll wieder in den zum Index gehörenden String umgewandelt. Andererseits wurden für manche Nachweise mehrere Funktionsaufrufe erstellt, welche für unterschiedliche Querschnittswerte gültig sind. Dies geschah beispielsweise für den Stahlbau für unterschiedliche I Profile.

In den folgenden Kapiteln werden die Realisierung der Anbindung der Funktionen sowie die Erweiterungsmöglichkeiten näher beschrieben.

### 1.6.1 Erstellen eigener Mathcad-Funktionen in C/C++ [14]

Mathcad bietet eine Schnittstelle für C, beziehungsweise C++, die es ermöglicht, eigene Funktionen in diesen Sprachen zu programmieren und an Mathcad anzubinden.

#### 1.6.1.1 Die Definition von DLL[26] für Mathcad

Dynamisch gelinkte Bibliotheken werden während des Ablaufs von Programmen dynamisch geladen. MATHCAD jedoch ruft die DLLs nicht dynamisch auf, sondern lädt sie bei Programmstart aus dem Verzeichnis \userefi in den Speicher [15].

#### 1.6.1.2 Programmierung einer DLL

Um eine DLL zu erstellen, muss zunächst eine Funktion in C/C++ geschrieben werden. MathSoft[27] schlägt hierzu folgende 32 – Bit – Compiler vor:

? Microsoft C/C++ Compiler
? Borland C/C++ 4.0 und 5.0
? Symantec C/C++ 6.0 und 7.0
? Watcom[28] C/C++ 9.5 und 10.0

[26] DLL – Dynamically Linked Library oder Dynamic Link Library.
[27] Firma MathSoft International, Knightway House, Park Street, Bagshot, Surrey, GU19 5AQ, United Kingdom

Compiler sind Systemprogramme, die den Quellcode eines Programms in Maschinencode umsetzen, der von einem Computer verstanden und ausgeführt werden kann. Die oben genannten Compiler wurden von MathSoft getestet. Bei Gebrauch anderer Compiler ist nicht garantiert, dass die Funktionen fehlerfrei geladen werden können. Die Empfehlung stammt zwar aus der aktuellen Version von Mathcad 2000, dennoch lagen die selben Empfehlungen mindestens seit Erscheinen der Version Mathcad 8 vor. Es sind also auch neuere Versionen der empfohlenen 16-Bit Compiler auf dem Markt erhältlich [16][17][18] oder der Vertrieb wurde mittlerweile eingestellt.

Im folgenden wird das Erstellen einer DLL mit der Entwicklungsumgebung von Microsoft Visual C++ 6.0 anhand eines einfachen Beispiels beschrieben. Die groben Schritte zur Erstellung einer DLL sind folgende[19]:

- ? Deklarierung der Benutzerfunktion
- ? FUNCTIONINFO – Struktur
- ? Fehlermeldungstabelle (optional)
- ? Definition der Benutzerfunktion, der ausführbare Quelltext
- ? DLL – Eintrittspunkt

Die Farbgebung im Quelltext ist vom Compiler zur besseren Übersicht eingerichtet. Grün geschrieben sind die Kommentare, blau steht für reservierte Befehle in C/C++ !

```
//*****************************************************************
// Diese Funktion übernimmt von MATHCAD zwei Argumente b und c, berechnet
// die Division b/c und übergibt das Ergebnis der Rückgabevariable a. Aufgerufen
// wird sie über den Funktionsnamen DIV(b, c).
//*****************************************************************

// mcadincl.h ist eine von MathSoft bereitgestellte sogenannte Header –
// Datei. Sie enthält die Protoypen für folgende Funktionen:
// CreateUserFunction, CreateUserMessageTable, MathcadAllocate, MathcadFree,
// MathcadArrayAllocate, MathcadArrayFree, MyCFunction und isUserInterrupted.
// Dies sind alles Funktionen für die Erstellung einer DLL.
// Weiterhin enthält die Datei die Typendefinitionen für die Strukturen
// COMPLEXSCALAR, COMPLEXARRAY und FUNCTIONINFO.

// Diese Einbindung ist immer notwendig!

#include "mcadincl.h"

//*****************************************************************
// Zu jedem C – Compiler[29], Bestandteil von, gehört eine Bibliothek, in der sich bereits
// übersetzte Funktionen befinden. In math.h befinden sich zahlreiche mathematische
```

[28] WATCOM C/C++ ist ein 80x86 basierter ANSI C++ und C Compiler. Er unterstützt viele Betriebssysteme. Die Firma WATCOM existiert derzeit nicht mehr. Der Compiler kann nicht mehr käuflich erworben werden und der Support endete am 30. Juni 2000.

[29] Bei der Erstellung des Programms fand Microsofts derzeit neuester C++ Compiler Verwendung. Dieser ist Bestandteil der Entwicklungsumgebung Visual C++ 6.0

```
// Standardfunktionen, wie cos(x), sqrt(x), und dergleichen [20].

#include "math.h"

//*******************************************************************
// Deklaration der Funktion DIV(b, c)
// LPCOMPLEXSCALAR a bedeutet, dass der Rückgabewert a (= Ergebnis)
// ein Skalarwert mit imaginärem Teil ist. Der imaginäre Teil ist immer dabei!
// Falls der Rückgabewert ein Feld ist, muss an dieser Stelle LPCOMPLEXARRAY
// a stehen. Das gleiche gilt für die Argumente b und c der Funktion DIV(b, c).

LRESULT DIVFunction(LPCOMPLEXARRAY a,
        LPCOMPLEXSCALAR b, LPCOMPLEXSCALAR c);

// Bei Registrierung mehrerer Funktionen in einer DLL folgen hier weitere
// Deklarationen.

//*******************************************************************
// Die FUNCTIONINFO – Struktur DIV wird für die Registrierung der
// Benutzerfunktion in MATHCAD verwendet. Sie enthält die Informationen über
// den Namen der Funktion, eine Beschreibung der Argumente, sowie den Zeiger
// auf den für die Ausführung der Berechnungen zuständigen Quelltext. Es muss für
// jede Funktion in der Bibliotheksdatei eine FUNCTIONINFO – Struktur angelegt
// werden.

// Die Informationen über die Funktion erscheinen im Dialogfeld Funktion einfügen.

FUNCTIONINFO DIV = {
        // Name, über welche die Funktion in MATHCAD aufgerufen wird
        "DIV",
        // Beschreibung der benötigten Funktionsparameter
        "a, b",
        // Beschreibung der Funktion, was liefert sie
        "Die Funktion DIV(a, b) liefert als "
        "Ergebnis die Division a durch b. ",
        // Zeiger auf den ausführbaren Quelltext, wenn der Benutzer den
        // Namen der Funktion eingibt
        (LPCFUNCTION)DIVFunction,
        // Rückgabewert ist ein komplexer Skalarwert. Falls der Rückgabewert ein
        // komplexes Feld ist, muss hier COMPLEX_ARRAY stehen!
        COMPLEX_SCALAR,
        // die Funktion benötigt/verwendet 2 Argumente
        2,
        // beide Argumente sind komplexe Skalare
{ COMPLEX_SCALAR, COMPLEX_SCALAR }
};

//*******************************************************************
```

```
// Rot umrandete Fehlermeldung in MATHCAD.
// Falls die Funktion niemals Fehler produziert, muss diese Tabelle nicht angelegt
// werden. In der eckigen Klammer steht die Anzahl der verschiedenen möglichen
// Fehlermeldungen.
// Pro DLL kann nur eine Fehlermeldungstabelle, aber mehrere Funktionen registriert
// werden. Alle Funktionen einer DLL greifen auf diese Tabelle zu.
// Die untere Fehlermeldungstabelle ist eigentlich überflüssig! MATHCAD erkennt
// folgende Ausnahmefehler: Überlauf, Division durch 0 und ungültige Operation und
// zeigt unter der jeweiligen Funktion die entsprechende Fehlermeldung an.

char *myErrorMessageTable[1] =

{
        "Division durch Null ist nicht erlaubt!"
};

//***************************************************************
// Der ausführbare Quelltext der Funktion DIV(a, b). Die Definition der Funktion.

LRESULT DIVFunction(LPCOMPLEXSCALAR a, LPCOMPLEXSCALAR b,
LPCOMPLEXSCALAR c)

{       // Die Daten zwischen MATHCAD und der Benutzerfunktion werden in
        // doppelter Genauigkeit ausgetauscht. Bei Verwendung unterschiedlicher
        // Datentypen sind entsprechende Konvertierungen zu beachten.

double DIVISION;
        double ARG1;
        double ARG2;

        // Übernahme der realen Teile der Argumente, falls die imaginären Teile
        // nicht benötigt werden, bzw. gesondert behandelt werden (müssen).

        ARG1 = b -> real;
        ARG2 = c -> real;

        // Wenn ARG2 = 0 ist, erscheint eine Fehlermeldung.
        // Im Fehlerfall setzt sich der Rückgabewert aus 2 Komponenten
        // zusammen: der Fehlercode und die Stelle (Argument) , an der die
        // Fehlermeldung erscheinen soll.
        // MAKELRESULT(1, 2) bedeutet Fehlercode 1 aus der Fehlermeldungs-
        // tabelle am 2. Argument.
        // Wenn die ganze Funktion rot markiert werden soll, weil z.B. ein
        // allgemeiner Fehler vorliegt, der keinem Argument zugeordnet werden
        // kann, lautet der Befehl: return 1.
        if ( ARG2 = = 0 )
                return MAKELRESULT (1, 2);
```

```
        // Berechnungen
        DIVISION = ARG1 / ARG2;

        // Rückgabe des Realteiles des Ergebnisses.
        // Wenn der Rückgabewert ein Feld ist, muss mit der Struktur
        // MathcadArrayAllocate Speicherplatz reserviert werden. Bei Rückgabe
        // einer Fehlermeldung muss dieser Speicherplatz mit MathcadArrayFree
        // wieder freigegeben werden. Daher empfiehlt es sich erst „kurz“ vor
        // Rückgabe des Ergebnisses, nach Überprüfung sämtlicher Fehlerquellen,
        // Speicherplatz zu reservieren! Zur Verwendung entsprechender Befehle
        // siehe Quelltext auf Seite 30.
         a -> real = DIVISION;

        // kein Fehler aufgetreten, normale Rückgabe
        return 0;
}
//*****************************************************************
// Alle MATHCAD – DLLs benötigen einen DLL – Eintrittspunkt.
// Der DLL – Eintrittspunkt wird durch das Betriebssystem beim Laden der DLL
// aufgerufen. Folgende Zeilen sind Microsoft spezifisch!

BOOL WINAPI _CRT_INIT(HINSTANCE hinstDLL, DWORD
        dwReason, LPVOID lpReserved);
BOOL WINAPI  DllEntryPoint (HINSTANCE hDLL, DWORD dwReason,
        LPVOID lpReserved)
// Statt “DllEntryPoint” kann jeder beliebige Begriff eingesetzt werden. Dies muss
// bei den Einstellungen des Linkers berücksichtigt werden, siehe hierzu Bild 43 !
{
        switch (dwReason)
        {
                case DLL_PROCESS_ATTACH:
                        if (!_CRT_INIT(hDLL, dwReason, lpReserved))
                                return FALSE;
                        // folgende 2 Zeilen sind wegzulassen, wenn keine
                        // Fehlermeldungstabelle angelegt wurde, die Funktion
                        // also keine Fehler produziert.
                        // Die Ziffer gibt die Anzahl der angelegten Fehlercodes
                        // an.
                        if (CreateUserErrorMessageTable(hDLL, 1,
                                        myErrorMessageTable))
                        // Registrierung der Benutzerfunktion(en)
                                CreateUserFunction( hDLL, &DIV );
                        break;
                case DLL_THREAD_ATTACH:
                case DLL_THREAD_DETACH:
                case DLL_PROCESS_DETACH:
                        if (!_CRT_INIT(hDLL, dwReason, lpReserved))
                                return FALSE;
```

```
                        break;
        }
        return TRUE;
}
//*********************************// Ende des Quelltextes in C/C++.
```

Programmlisting 4.

Dieser Quelltext für die Erstellung einer DLL stellt das Grundgerüst dar. Durch Änderungen des ausführbaren Quelltextes, der Definition der Funktion und Anpassungen der notwendigen Strukturen, insbesondere den Zeigern auf den ausführbaren Quelltext, kann dieses Kapitel als Leitfaden zur Erstellung eigener DLLs benutzt werden.

### 1.6.1.3 Änderungen für C++

Die obige Beschreibung und die von MathSoft mitgelieferte Datei *mcadincl.h* lassen Quelltexte nur in C zu. Um C++ verwenden zu können sind einige Änderungen notwendig.

Im Abschnitt des DLL – Eintrittpunktes muss die Zeile

```
BOOL WINAPI _CRT_INIT(HINSTANCE hinstDLL, DWORD dwReason,
        LPVOID lpReserved);
```

geändert werden in

```
extern "C" BOOL WINAPI _CRT_INIT(HINSTANCE hinstDLL, DWORD
        dwReason, LPVOID lpReserved);
```

Für ein fehlerfreies Linken (s. Kapitel 1.6.1.4) sind noch Änderungen in der Datei *mcadincl.h* vorzunehmen. Den fünf aufgelisteten Zeilen wurde jeweils

```
extern "C"
```

vorangestellt.

```
extern "C" const void * CreateUserFunction( HINSTANCE, FUNCTIONINFO * );

extern "C" BOOL CreateUserErrorMessageTable( HINSTANCE, …

extern "C" BOOL MathcadArrayAllocate( COMPLEXARRAY * const, …

extern "C" void MathcadArrayFree( COMPLEXARRAY * const );

extern "C" BOOL isUserInterrupted( void );
```

Das Linken erfolgt danach fehlerfrei, auch mit C++ Befehlen im Quelltext.

#### 1.6.1.4 Einstellungen des Compilers

Mit dem nach Kapitel 1.6.1.2 erstellten Quellcode und den nach Kapitel 1.6.1.3 durchgeführten Änderungen kann nun eine DLL erstellt werden. Die DLL wird vom Compiler erstellt. Hierfür müssen folgende Einstellungen des Compilers vorgenommen werden [21]:

1. Schritt: In der Menüleiste der Entwicklungsumgebung von Microsoft Visual C++ unter dem Menüpunkt Datei *Neu...* wählen. Es erscheint:

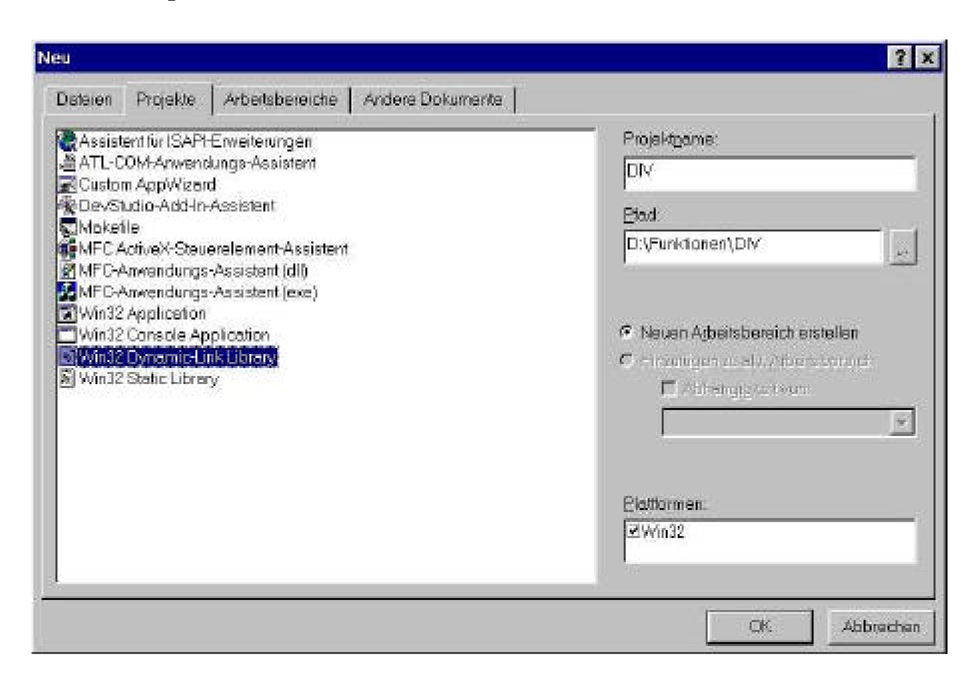



Bild 38. Menuefenster *Datei/Neu...* der Entwicklungsumgebung von MS Visual C++

Im Register *Projekte* „Win32 Dynamic – Link Library“ wählen.
Unter *Projektname* und *Pfad* Entsprechendes eingeben!

**2. Schritt:** Nach Bestätigung erscheint der Projektassistent der Entwicklungsumgebung, wie Bild 39 zeigt.

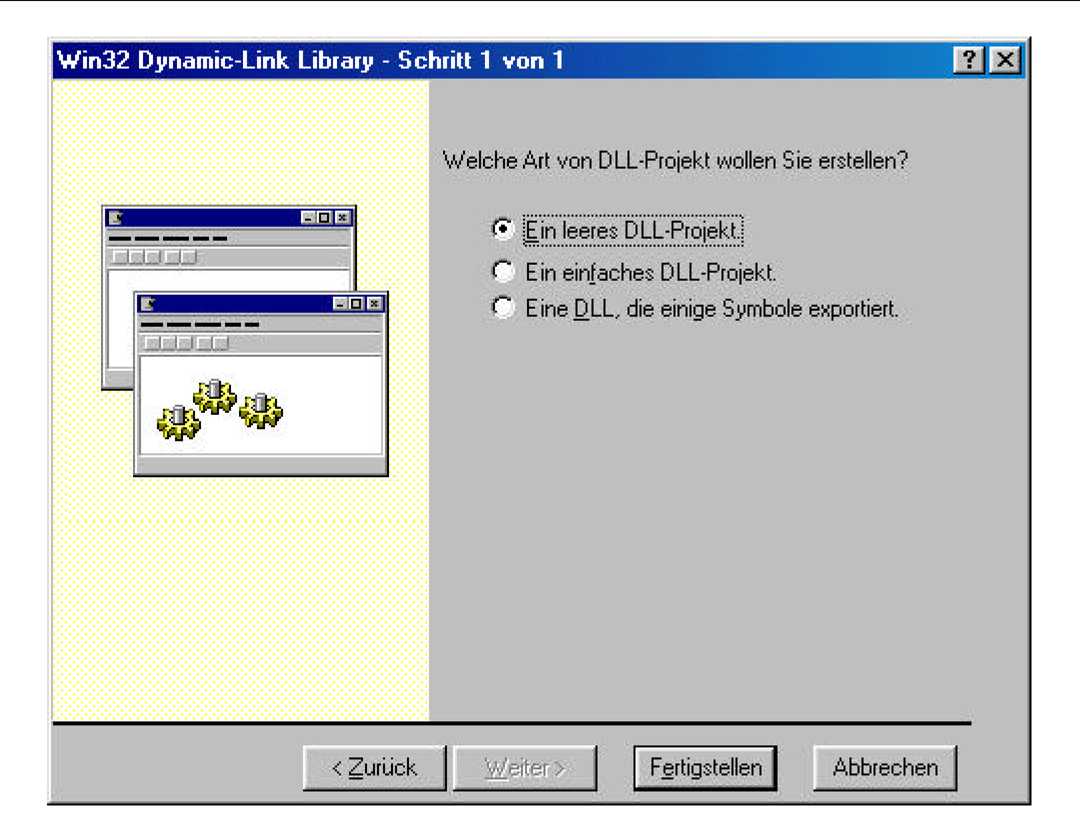



Bild 39. Projektassistent Schritt 1 der Entwicklungsumgebung von MS Visual C++

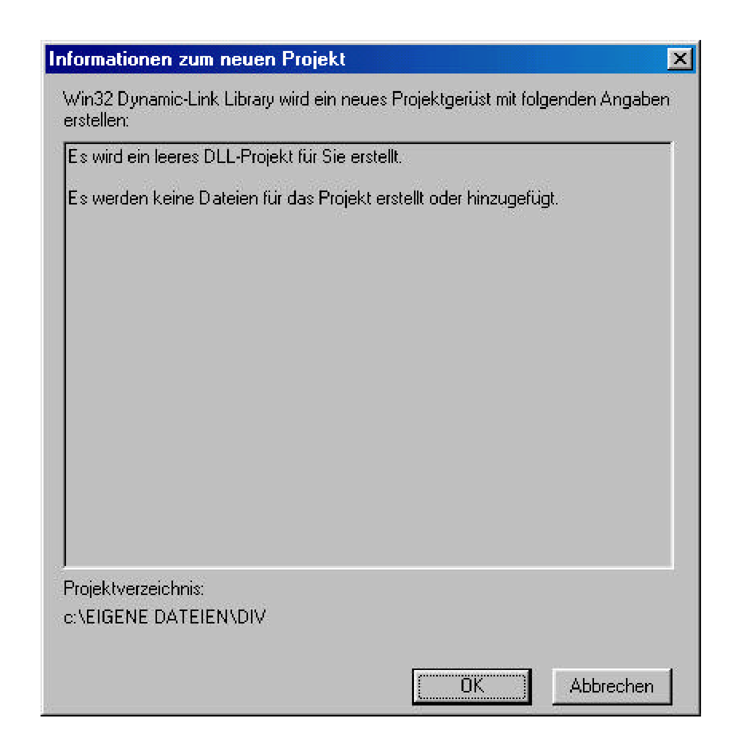



Bild 40. Informationsfenster der Entwicklungsumgebung MS Visual C++

Nach dem Klicken auf *Fertigstellen* erscheint ein Informationsfenster, wie Bild 40 zeigt, das Angaben über das soeben erstellte Projekt enthält, wie beispielsweise den Verzeichnispfad, unter dem das Projekt gespeichert wurde oder welche Dateien dem Projekt hinterlegt wurden.




Bild 41. Projektansicht der Entwicklungsumgebung Microsoft Visual C++

Obiges Bild 41 zeigt schließlich die Entwicklungsumgebung in der Dateiansicht. Die Ordner für die unterschiedlichen Dateitypen, wie Header-Dateien mit der Endung .h oder die Implementierungsdateien mit den Endungen .cpp werden vom Programm bereits selbstständig hinzugefügt.

Mit Hilfe des Kontextmenüs (rechte Maustaste) können dann beispielsweise folgende Dateien zum Projekt DIV und den entsprechenden Ordnern hinzugefügt werden:

Quelltext DIV.cpp

Header – Datei mcadincl.h

Die Datei befindet sich bei Verwendung des Microsoft Visual C++ Compilers im Verzeichnis Mathsoft\Mathcad\userefi\Microsft\Include[30]

Bibliotheksdatei mcaduser.lib im Verzeichnis
Mathsoft\Mathcad2000\userefi\Microsft\Lib

Bemerkungen: Die Header – Datei *math.h* ist nur hinzuzufügen, wenn sie im Quelltext eingebunden wurde.

**3. Schritt:** Im Kontextmenü zu DIV Dateien erscheint unter *Einstellungen* das Dialogfenster *Projekteinstellungen*:

[30] Der Name „Microsft“ im Installationspfad ist kein Schreibfehler und wurde vom Setup so installiert

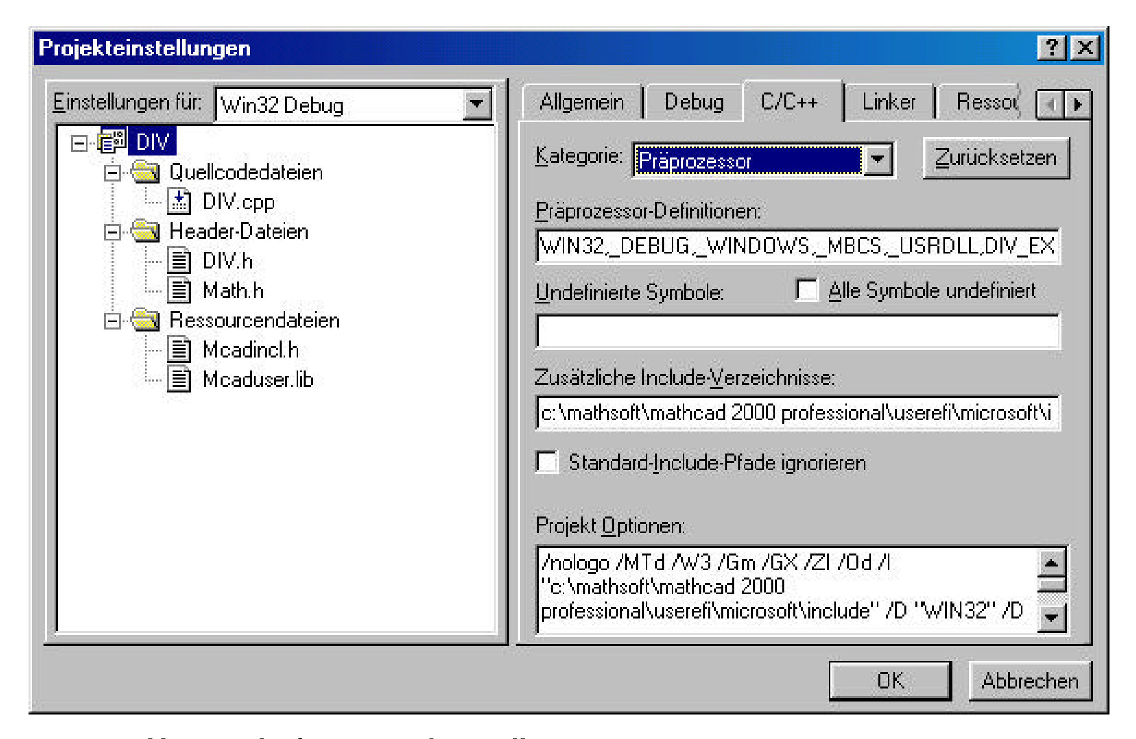



Bild 42. Dialogfenster Projekteinstellungen

Hier können unter der Registerkarte *C/C++* verschiedene Einstellungen gewählt werden. Einsztellungen sollten wie folgt gewählt werden:

- Kategorie *Präprozessor* einstellen.

Die Header – Datei *mcadincl.h* ist nicht in der Bibliothek des Compilers enthalten ( sie wird von MathSoft geliefert! ). Der zugehörige Pfad ist unter *Zusätzliche Include – Verzeichnisse* einzugeben.

**4. Schritt:** Im Register *Linker* ist der DLL – Eintrittspunkt einzugeben.

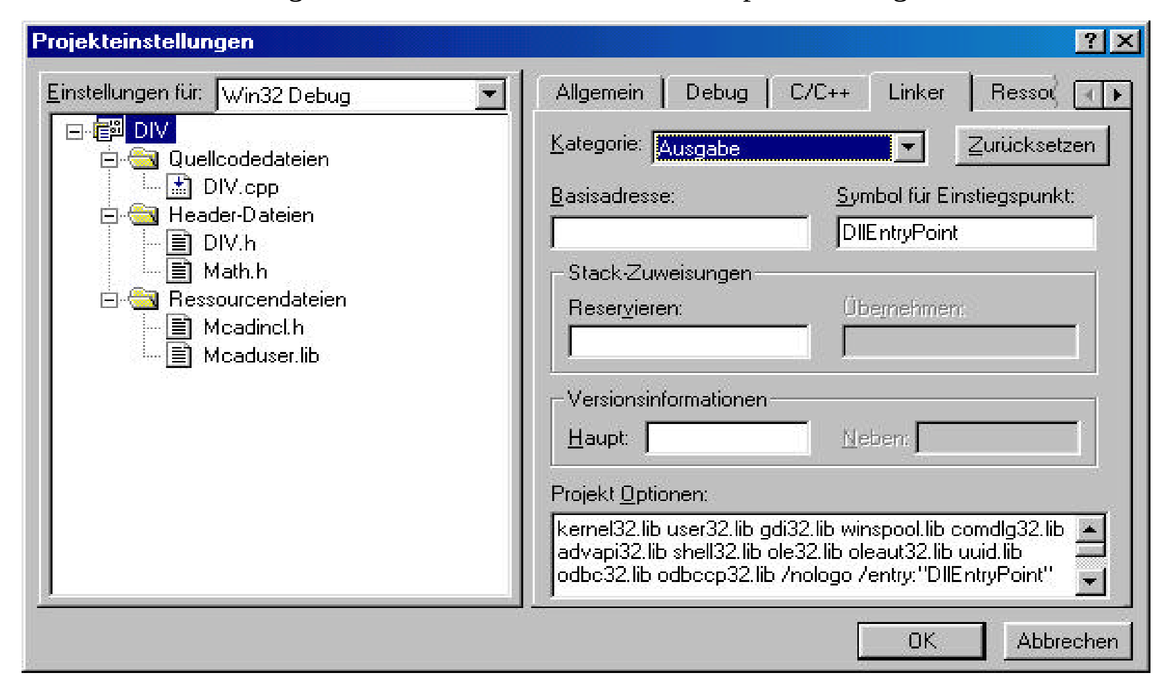



Bild 43. Einstellungen auf dem Tabellenblatt Projekteinstellungen/Linker

Hierzu wird die Kategorie *Ausgabe* ausgewählt und bei *Symbol für Einstiegspunkt* der im Quelltext gewählte Begriff eingetragen, hier: „DllEntryPoint".

**5. Schritt:** In der Menüleiste der Entwicklungsumgebung kann nun unter dem Menüpunkt *Erstellen, DIV.dll erstellen* gewählt werden um die Anwendung im angegebenen Pfad zu kompilieren.

#### 1.6.1.5 Einbindung der Benutzerfunktionen in MATHCAD

Beim Programmstart sucht MATHCAD vorab im Verzeichnis \userefi nach Dateien mit einer .dll – Erweiterung. Danach versucht das Programm alle Dateien mit dieser Endung zu laden.

Nach Erstellung der DLL muss diese also in das Verzeichnis \userefi kopiert werden, um sie in Mathcad verfügbar zu machen. Die Benutzerfunktion erscheint dann wie die integrierten Funktionen im Dialogfenster *Funktion einfügen*.

Beim Erstellen und Testen der DLLs wurde in Version Mathcad 8 festgestellt, dass die Obergrenze der Dateien mit .dll – Erweiterungen bei 50 liegt. Sind mehr als 50 DLLs im Verzeichnis \userefi vorhanden, ließ sich das Programm nicht mehr starten. Diese Beschränkung ist in der Version Mathcad 2000 offenbar aufgehoben worden, wie ein neuerer Test ergab .

Bei der Erstellung einer Dll ist zu beachten, dass nur eine Fehlermeldungstabelle angelegt werden kann. Zu jeder Funktion in einer DLL muss eine Deklarierung, eine FUNCTIONINFO – Struktur und der ausführbare Quelltext vorhanden sein.

Die Registrierung der einzelnen Funktionen in der DLL – Eintrittsroutine muss als Block erfolgen.

```
…
if (CreateUserErrorMessageTable(hDLL, 1, myErrorMessageTable))

// Registrierung der Benutzerfunktion(en)
        {
                CreateUserFunction( hDLL, &Funktion1 );
                CreateUserFunction( hDLL, &Funktion2 );
                …
        }
break;
…
```

Programmlisting 5.

### 1.6.2 Verwendung der Funktionen in Mathcad [8]

Die nach Kapitel 1.6.1 erstellten Dll's können schließlich an MATHCAD angebunden werden. Die Registrierung der erstellten Funktionen erfolgt mittels einer xml Datei[31]. Siehe hierzu auch [22].

Der Aufruf der Funktionen erfolgt in Mathcad über das Dialogfensters "Funktion einfügen". Die Informationen über die Funktionen können über die xml Datei übersichtlich aufgebaut werden. Sie können beispielsweise in Kategorien eingeteilt werden und erscheinen damit, thematisch getrennt voneinander im Fenster. Bild 44 zeigt dies im Fall von Funktionen des Stahlbetonbaus.

Wie in Kapitel 1.5 erwähnt stellt Mathcad innerhalb der XML-Datei unter der Rubrik <description> die Möglichkeit zur Verfügung, Informationen oder eine Eingabehilfe für die ausgewählte Funktion bereitzustellen. Diese Informationen erscheinen dann in einem kleinen Anzeigefeld innerhalb des Dialogfensters *Funktion Einfügen*.

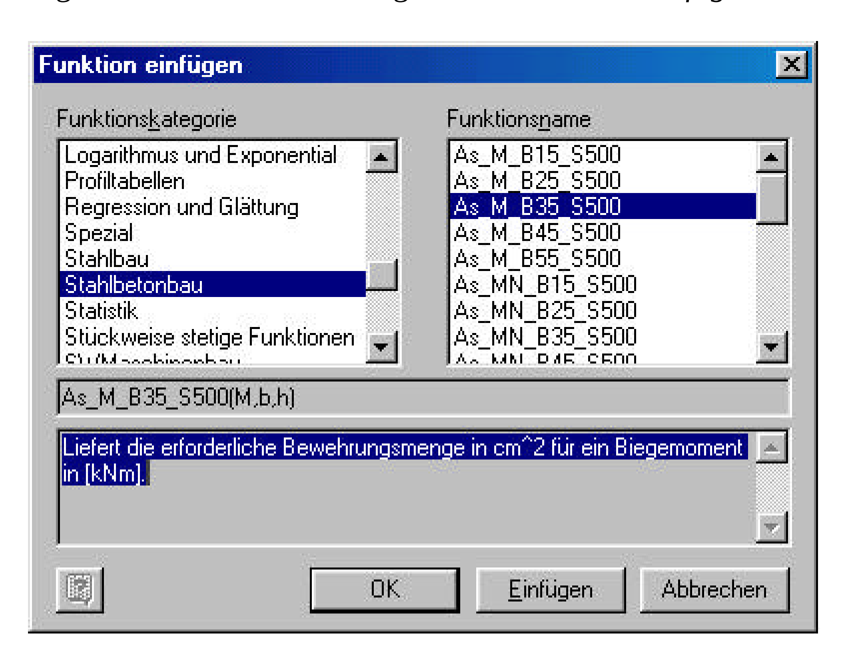



Bild 44. Das *Funktion einfügen* Dialogfenster von Mathcad 2000

Nachfolgendes Programmlisting zeigt einen Ausschnitt aus der Datei, welche die Funktion As_M_B35_S500 in Mathcad verfügbar macht.

```
<?XML version="1.0" encoding="UTF-8" ?>
<!DOCTYPE FUNCTIONS SYSTEM "functions.dtd">

<FUNCTIONS>
...
<function>

<name>
As_M_B35_S500
```

[31] XML - Extensible Markup Language. XML wurde aus der textorientierten Sprache HTML und der Idee entwickelt, ein professionelles, graphikorientiertes Layout für das Web Design zu erhalten. Siehe hierzu auch

```
</name>

<params>
M,b,h
</params>

<category>
Stahlbetonbau
</category>

<description>
Liefert die erforderliche Bewehrungsmenge in cm ^ 2 für ein Biegemoment in [kNm].
</description>

<help_topic>
</help_topic>
</function>
…
</FUNCTIONS>
```

Programmlisting 6.

# Literatur

[1] Kofler, Michael
Visual Basic 6
Verlag Addison-Wesley-Longman; Bonn; 1998

[2] Goll/Weiß/Rothländer
Java als erste Programmiersprache
B.G.Teubner Stuttgart , Leipzig 2000

[3] Sun Microsystems GmbH
Zettachring 10a
70567 Stuttgart
Deutschland
Tel: 0711-72098-0, Fax: 0711-72098-43
http://www.sun.de

[4] Visual J++ reader poll
Online Umfrage
Jenni Aloi
Mai 1998
http://www.javaworld.com/javaworld/jw-05-1998/jw-05-pollresults.html

[5] MSDN Library
Microsoft Developer Network
Online Dokumentation zu Microsofts Entwicklungsumgebungen
Visual Studio 6.0-Release, Aktualisiert alle drei Monate

[6] JavaScript
Online Veröffentlichung
Dietmar Rabich
Dülmen, Januar 1999
http://rabich.de/expertenforum/javascript_pro_und_contra/index.htm

[7] Mathcad deutsche Maschinenbau-Bibliothek
Software für den Maschienenbau, Dezember 2000
Softwert GmbH
Am Meerkamp 21
D-40667 Meerbusch
http://www.softwert.com

[8] Datenbanken und SQL
Edwin Schicker
Eine praxisorientierte Einführung mit Hinweisen zu Oracle und MS-Access.
B.G. Teubner Verlag; 3. Auflage 2000

[9] Jochen Günther; Karsten Brugger
Softwareentwicklung zur Anwendung der relationalen Datenbank MS-Access
im Konstruktiven Ingenieurbau, Diplomarbeit im Fachbereich Bauingenieurwesen
Referent: Prof. Dr. H. Werkle, Fachhochschule Konstanz, Januar 2001

[10] Said Baloui,
Access 2000 Kompendium
Datenbanken planen, entwickeln, optimieren
Markt + Technik Verlag; München; 2000

[11] WexTech Systems, Inc.
400 Columbus Avenue
Valhalla, NY 10595
United Staates
http://www.wextech.com/

[12] Software Interphase, Inc.
82 Cucumber Hill Road
Foster, RI 02825-1212
USA
http://www.sinterphase.com

[13] eHelp Corporation
10590 W. Ocean Air Drive
San Diego, CA 92130
USA
http://www.ehelp.com

[14] Serkan Karadag
Statische Berechnungen mit Mathcad als Engineering Desktop Tool
Diplomarbeit im Fachbereich Bauingenieurwesen, Referent: Prof. Dr. H. Werkle
Fachhochschule Konstanz, Februar 2000

[15] MATHCAD 2000 Benutzerhandbuch
MathSoft Inc.
Cambridge MA
USA
http://www.mathsoft.com

[16] Microsoft GmbH
Konrad-Zuse-Straße 1
85716 Unterschleißheim
Deutschland
http://www.eu.microsoft.com/germany

[17] Borland Software Corporation
Robert-Bosch-Straße 11
63225 Langen
Deutschland
http://www.borland.de

[18] Symantec Corporation
20330 Stevens Creek Blvd.
Cupertino, CA 95014
United States
www.symantec.com

[19] MATHCAD 2000
UserDLL.pdf
Acrobat Reader Datei, Bestandteil der CD für die Programminstallation
Version 2000, Erscheinungsjahr 1999

[20] MICROSOFT VISUAL C++
Entwicklungsumgebung
Version 6.0
Microsoft GmbH

[21] Readme - Datei
im Verzeichnis ...Mathsoft\Mathcad2000\userefi\Microsft
im Installationspfad des Programms MATHCAD 2000

[22] Das Web automatisieren mit XML
Online Dokumentation
Kuno Dunhoelter
Version 2.0, 1. September 1998)
http://members.aol.com/xmldoku/

# II-2 Softwarekomponenten

## 2.1 Allgemeines

Dieses Kapitel befasst sich mit den baustatischen Grundlagen der Softwarekomponenten sowie mit deren objektorientierter Analyse für die Programmiersprache Java. Weiterhin wird die Ein- und Ausgabe im Rahmen des Engineering Desktop kurz skizziert.

## 2.2 Allgemeiner Träger

Unter einem allgemeinen Träger wird ein beliebig gelagerter und belasteter Biegebalken verstanden, der folgende Eigenschaften besitzt:

- Einschluss von Schubverformungen
- Elastische Bettung
- Theorie I-ter und II-ter Ordung

Hierunter fallen Einfeldträger, Durchlaufträger, elastisch gebettete Träger und ein- oder mehrfeldrige Stützen nach Theorie II-ter Ordnung.

### 2.2.1 Theoretische Grundlagen

Zur Berechnung einfacher Einfeld- und Durchlaufträger gibt es eine Reihe statischer Verfahren. Eine systematische Vorgehensweise, die insbesondere für die Programmierung geeignet ist, stellt das Übertragungsmatrizenverfahren dar [1-3]. Hierfür wurde ein Klassenmodell entwickelt und in JAVA implementiert.

#### 2.2.1.1 Berechnungsverfahren

Dem Rechenkern des Durchlaufträger-Moduls liegt das Übertragungsmatrizenverfahren mit der speziellen Formulierung der Übertragungsmatrix nach Rubin [4] zugrunde.

Das Verfahren der Übertragungsmatrizen ist ein elektronisches Berechnungsverfahren, das speziell zur Ermittlung der Schnittgrößen und Einflusslinien von Durchlaufträgern entwickelt wurde. Es geht auf die Arbeiten von Falk [1] zurück. Eine zusammenfassende Darstellung findet sich u.a. in [2] sowie in der hier angewandten Formulierung in [3].

Das Übertragungsmatrizenverfahren zeichnet sich durch seinen leicht schematisierbaren Rechenablauf und durch die geringe Größe des zu lösenden Gleichungssystems aus. Diese ist unabhängig von der Größe des statischen Systems.

Der grundlegende Rechengang wird zunächst an einem Einfeldträger erläutert. Die Lagerbedingungen am linken und rechten Trägerrand sind beliebig. Weiterhin kann er auch mehrere Verschiebungs- und Drehfedern enthalten. Als Einwirkungen sind beliebige Einzel- und Streckenlasten möglich, Bild 45.

Der Trägers wird zur Berechnung in Abschnitte und Knotenpunkte unterteilt, wobei zwischen zwei Abschnitten jeweils ein Knotenpunkt liegt. Die Trägerabschnitte können durch

trapezförmige Streckenlasten belastet werden. An den Knotenpunkten können Einzelkräfte und –momente wirken. Weiterhin sind an den Knotenpunkten auch Federn möglich.

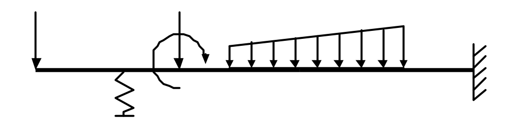


Bild 45. Berechung eines Trägers nach dem Übertragungsmatrizenverfahren

### 2.2.1.2 Zustandsvektor

Grundlegende Objekte des Übertragungsmatrizenverfahrens sind die Zustandsvektoren. Ein Zustandsvektor fasst die statischen Größen

Durchbiegung *w*
Drehwinkel ϕ
Moment *M*
Querkraft *Q*

$$\underline{Z} = \begin{bmatrix} w \\ f \\ M \\ Q \end{bmatrix}$$

an den Abschnittsgrenzen zusammen. An jedem Abschnitt und jedem Knotenpunkt gibt es demnach zwei Zustandsvektoren, nämlich jeweils am linken und rechten Rand.

### 2.2.1.3 Feldmatrix

Der Rechenablauf des Übertragungsmatrizenverfahrens beruht auf der „Übertragung“ des Zustandsvektors über den Träger. Man beginnt am linken Trägerrand und „überträgt“ den Zustandsvektor über die vorgegebene Abfolge von Abschnitten und Knotenpunkten bis zum rechten Rand. Hierzu benötigt man die Methoden, um einen Zustandsvektor über einen Abschnitt und einen Punkt zu übertragen. Diese sollen als Nächstes behandelt werden.

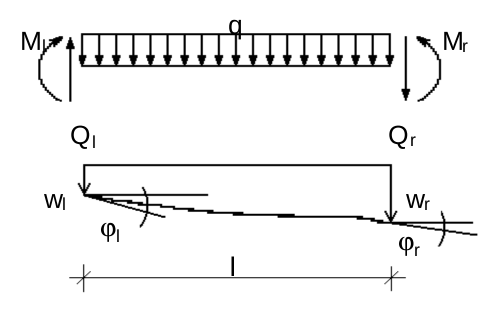



Bild 46. Trägerabschnitt

Die Beziehung zur Übertragung über einen Abschnitt ergibt sich aus der Differentialgleichung des geraden Biegebalkens mit konstanter Biegesteifigkeit $B = E\,I$ und Schubsteifigkeit $S = G\,A_Q$ zu[1]:

$$w^{IV} - K_1 \cdot w'' - K_2 \cdot w = -g\,\frac{d}{B}\sum_{0} a_j \cdot \bar{q}_j - \frac{g}{S}\sum_{2} a_{j-2} \cdot \bar{q}_j \qquad \text{Gl. (1)}$$

[1] Gleichung 3.2-24 aus [10]Seite 87

mit $$g = \frac{1}{1 - \frac{N}{S}}$$ Gl. (1a)

$$d = 1 - \frac{I_{\aleph} \cdot w^2}{S}$$ Gl. (1b)

$$J = g \cdot N_r + I_{\aleph} \cdot w^2$$ Gl. (1c)

$$K_1 = -\frac{J}{B} + g \frac{c_r}{S}$$ Gl. (1d)

$$K_2 = -g \cdot d \frac{c_r}{B}$$ Gl. (1e)

$$\overline{q}_j = q_j - N \cdot w^{V}_{j+2}$$ Gl. (1f)

Als Randbedingungen setzt man die Elemente $w_l, f_l, M_l$ und $Q_l$ des Zustandsvektors am linken Abschnittsrand ein. Die Elemente des Zustandsvektors am rechten Abschnittsrand erhält man aus der Lösung der Differentialgleichung des elastisch gebetteten schubweichen Biegebalkens nach Theeorie II-ter Ordnung. Hierzu wird die in [4] angegebene allgemeine Lösung mit Hilfe spezieller Reihen nach Rubin verwendet.

Die numerische Berechnung der Reihen-Beiwerte aj und bj ist in den Struktogrammen in Bild 2 und Bild 3 wiedergegeben.

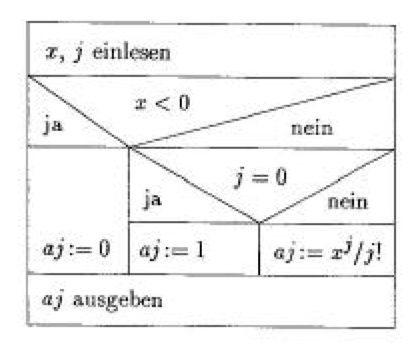



Bild 47. Struktogramm für die Beiwerte aj (siehe [4] Seite 140)

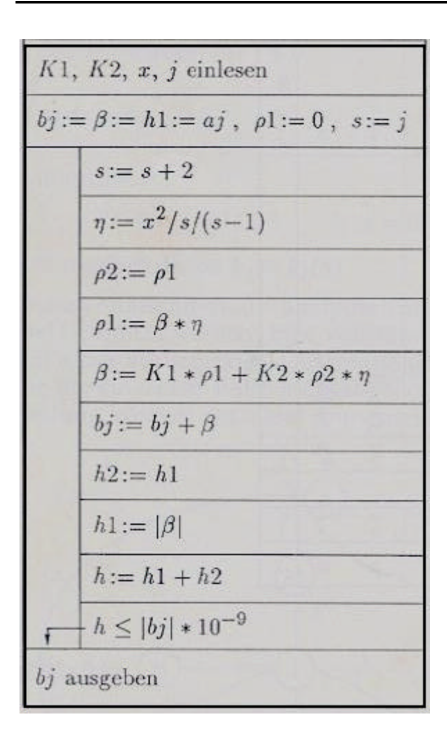



Bild 48. Struktogramm für die Beiwerte bj (siehe [4] Seite 140)

In den $b_j$ Werten sind bereits die Bettungsanteile sowie die Schubsteifigkeit mit eingearbeitet.

Die Übertragungsbeziehung lautet damit:

$$\begin{bmatrix} w_r \\ j_r \\ M_r \\ Q_r \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} \cdot \begin{bmatrix} w_\ell \\ j_\ell \\ M_\ell \\ Q_\ell \end{bmatrix} + \begin{bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \end{bmatrix} \qquad \text{Gl. (2)}$$

oder

$$\underline{Z}_r = \underline{A}_{Ab} \cdot \underline{Z}_l + \underline{L}_{Ab} \qquad \text{Gl. (3)}$$

Die Berechnung der Koeffizientenmatrix $\underline{A}_{Ab}$ und des Lastvektors $\underline{L}_{Ab}$ erfolgt mit Hilfe der o.a. Beiwerte. Sie lauten: nach [4] Seite 141:

$$k_{11} = b_0 + \frac{v}{B} \cdot b_2 \qquad k_{12} = g \cdot b_1 \qquad k_{13} = -g/B \cdot b_2 \qquad k_{14} = -g \cdot \left( \frac{1}{B} \cdot b_3 - \frac{1}{S} \cdot b_1 \right)$$

$$k_{21} = -g \cdot \frac{c_S}{B} \cdot b_3 \qquad k_{22} = b_0 - g \cdot \frac{c_S}{S} \cdot b_2 \qquad k_{23} = -\frac{1}{B} \cdot \left( b_1 - g \cdot \frac{c_S}{S} \cdot b_3 \right) \qquad k_{24} = -\frac{g}{B} \cdot b_2$$

$$k_{31} = g \cdot c_S \cdot b_2 \qquad k_{32} = v \cdot b_1 + g \cdot c_S \cdot b_3 \qquad k_{33} = b_0 - g \cdot \frac{c_S}{S} \cdot b_2 \qquad k_{34} = g \cdot b_1$$

$$k_{41} = c_S \cdot \left( b_1 + \frac{v}{B} \cdot b_3 \right) \qquad k_{42} = g \cdot c_S \cdot b_2 \qquad k_{43} = -g \cdot \frac{c_S}{B} \cdot b_3 \qquad k_{44} = b_0 + \frac{v}{B} \cdot b_2$$

mit:

$B = E \cdot I$ $\qquad S = G \cdot A_Q$ $\qquad c_S$: Bettungsmodul

$$g = \frac{1}{1 - N/S} \qquad v = g \cdot N \qquad \text{Gl. (4)}$$

und $$K1 = -\frac{v}{B} \cdot g \cdot \frac{c_S}{S} \qquad K2 = -g \cdot \frac{c_S}{B} \qquad \text{Gl. (5)}$$

für die Berechnung der Beiwerte $b_i$ nach Bild 4.

### 2.2.1.4 Punktmatrix

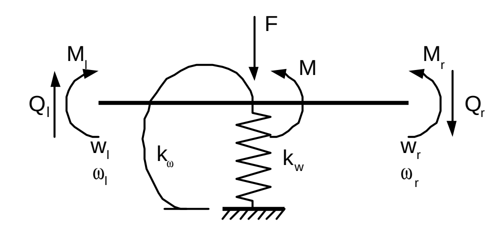



Bild 49. Knotenpunkt zwischen zwei Trägerabschnitten mit Senk- und Drehfeder

$$\Delta j_i = M_i / k_M$$
$$j_{ir} = j_{il} - M_i / k_M \qquad \text{Gl. (6)}$$

Bild 50. Knotenpunkt mit Relativverdrehungsfeder

$$\Delta w_i = Q_i / k_Q$$
$$w_{ir} = w_{il} + Q_i / k_Q \qquad \text{Gl. (7)}$$

Bild 51. Knotenpunkt mit Relativverschiebungsfeder

Zur Übertragung über einen Knotenpunkt betrachtet man einen Punkt, an dem eine Verschiebungs- und eine Drehfeder angreifen und der durch eine Einzellast und ein Moment belastet ist, Bild 49. Die Federkonstanten der Verschiebungsfeder wird mit $k_w$, diejenige der Drehfeder mit $k_f$ bezeichnet. Aus den Federgleichungen und den Gleichgewichtsbedingungen erhält man die Übertragungsbeziehung zu:

$$\begin{bmatrix} w_r \\ j_r \\ M_r \\ Q_r \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1/k_Q \\ 0 & 1 & -1/k_M & 0 \\ 0 & -k_j & 1 & 0 \\ k_w & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_\ell \\ j_\ell \\ M_\ell \\ Q_\ell \end{bmatrix} + \begin{bmatrix} w_P \\ j_P \\ -M_P \\ -F_P \end{bmatrix} \qquad \text{Gl. (8)}$$

oder

$$\underline{Z}_r = \underline{A}_{Pu} \cdot \underline{Z}_l + \underline{L}_{Pu} \qquad \text{Gl. (9)}$$

Die Punktmatrix $\underline{A}_{Pu}$ enthält die Federkonstanten. Der Lastvektor $\underline{L}_{Pu}$ beschreibt neben einer Einzelkraft und einem eingeprägten Moment am betrachteten Punkt auch eine eingeprägte Verschiebung und eine eingeprägte Verdrehungen. Damit können Auflagersenkungen und -verdrehungen untersucht werden.

#### 2.2.1.5 Lastvektor

Für die Berechnung stehen neben möglichen Einzellasten, Verschiebungen und Verdrehungen zusätzlich noch Strecken- Temperaturlasten zur Verfügung.

Diese sind im Lastvektor $L = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix}$ beinhaltet. Gl. (10)

Mit den Parametern $c_S$ = Bettungsmodul, B = Biegesteifigkeit $E \cdot I$, S = Schubsteifigkeit $G \cdot A_Q$, folgt unter Verwendung der Struktogramms nach Bild 4, mit den Eingangswerten nach den Gleichungen (4) und (5) ergeben sich für eine Trapezlast mit den Parametern $q_{li}$ und $q_{re}$ die Koeffizienten:

$$
\begin{aligned}
k_1 &= g \cdot \left(\frac{b_4}{B} - \frac{b_2}{S}\right) \cdot q_{li} + g \cdot \left(\frac{b_5}{B} - \frac{b_3}{S}\right) \cdot \frac{q_{re} - q_{li}}{x} \\
k_2 &= g \cdot \frac{b_3}{B} \cdot q_{li} + g \cdot \frac{b_3}{B} \cdot \frac{q_{re} - q_{li}}{x} \\
k_3 &= -g \cdot b_2 \cdot q_{li} + g \cdot b_3 \cdot \frac{q_{re} - q_{li}}{x} \\
k_4 &= -x \cdot q_{li} - 0.5 \cdot x^2 \cdot \frac{q_{re} - q_{li}}{x}
\end{aligned}
\qquad \text{Gl. (11)}
$$

Für eine aufgebrachte ungleichmäßige Temperaturlast $DT$ über die Querschnittshöhe $h$ mit dem materialspezifischen Temperaturausdehnungskoeffizient $a_T$ ergeben sich die Koeffizienten zu:

$$
\begin{aligned}
k_1 &= -g \cdot b_0 \cdot \frac{\Delta T}{h} \cdot a_T \\
k_2 &= -b_0 \cdot \frac{\Delta T}{h} \cdot a_T \\
k_3 &= -g \cdot N \cdot b_0 \cdot \frac{\Delta T}{h} \cdot a_T \\
k_4 &= 0
\end{aligned}
\qquad \text{Gl. (12)}
$$

#### 2.2.1.6 Berechnung von Einfeldträgern

Nach dem Aufstellen der beiden Beziehungen (Gl.2) und (Gl.8) für die Abschnitte, Unterabschnitte und Punkte, kann mit der Übertragung des Zustandsvektors am linken Trägerrand begonnen werden. Dieser ist aber, da es sich bei der Differentialgleichung (Gl.1) um ein Randwertproblem und nicht um ein Anfangswertproblem handelt, nicht vollständig bekannt. Vielmehr sind von den vier Elementen des Zustandsvektors sowohl am linken wie am rechten Trägerrand jeweils zwei Elemente bekannt und zwei unbekannt. Fasst man die

unbekannten Größen zu einem Vektor zusammen, so kann der Zustandsvektor am Trägeranfang nach Tabelle 5 geschrieben werden:

$$\underline{Z}_A = \underline{U}_A \cdot \underline{X} \qquad \text{Gl. (13)}$$

Allgemein hat der Zustandsvektor die Form:

$$\underline{Z} = \underline{U} \cdot \underline{X} + \underline{L} \qquad \text{Gl. (14)}$$

Hierin bedeuten $\underline{U}$ die von den Unbekannten $\underline{X}$ abhängigen Terme des Zustandsvektors und $\underline{L}$ die zugehörigen Lastterme, die am Trägeranfang zunächst Null sind. Der Zustandsvektor wird nun in dieser Form über die Abschnitte und Punkte übertragen. Mit dem Zustandsvektor

$$\underline{Z}_l = \underline{U}_l \cdot \underline{X} + \underline{L}_l \qquad \text{Gl. (15)}$$

am linken Rand und der Übertragungsvorschrift

$$\underline{Z}_r = \underline{A} \cdot \underline{Z}_l + \underline{L}_A \qquad \text{Gl. (16)}$$

die beim Abschnitt (Gl.3) bzw. beim Knotenpunkt (Gl.9) entspricht, erhält man

$$\underline{Z}_r = \underline{A} \cdot \left(\underline{U}_l \cdot \underline{X} + \underline{L}_l\right) + \underline{L}_A \qquad \text{Gl. (17)}$$

beziehungsweise

$$\underline{Z}_r = \left(\underline{A} \cdot \underline{U}_l\right) \cdot \underline{X} + \left(\underline{A} \cdot \underline{L} + \underline{L}_A\right) \qquad \text{Gl. (18)}$$

$$\underline{Z}_r = \underline{U}_r \cdot \underline{X} + \underline{L}_r \qquad \text{Gl. (19)}$$

oder mit

$$\underline{U}_r = \underline{A} \cdot \underline{U}_l \qquad \text{Gl. (20)}$$

und

$$\underline{L}_r = \underline{A} \cdot \underline{L} + \underline{L}_A \qquad \text{Gl. (21)}$$

Der mit Hilfe von Tabelle 5 aufgestellte Anfangsvektor wird mit(Gl.19), (Gl.20) und (Gl.21) über die einzelnen Abschnitte und Punkte des Trägers bis zu dessen rechtem Ende übertragen. Dabei sind die Zustandsvektoren am rechten Rand eines Abschnittes und am linken Rand des darauffolgenden Punktes jeweils gleich. Wirken an dem Punkt zwischen zwei Abschnitten keine Einzellasten bzw. -momente und sind keine Federn vorhanden, kann auch direkt zum nächsten Abschnitt übergegangen werden. Am Trägerende sind ähnlich wie am Trägeranfang wieder zwei Lagerbedingungen bekannt und zwei weitere unbekannt. Ähnlich wie am Trägeranfang lassen sich also die Lagerbedingungen am Trägerende schreiben:

$$\underline{Z}_E = \underline{U}_E \cdot \underline{Y} \qquad \text{Gl. (22)}$$

wobei $\underline{Y}$ die Unbekannten am Trägerende bezeichnet und $\underline{U}_E$ mit Tabelle 5 bestimmt wird. Dieser Zustandsvektor ist gleich dem durch die Übertragung erhaltenen Zustandsvektor,

$$\underline{Z} = \underline{U} \cdot \underline{X} + \underline{L} \qquad \text{Gl. (23)}$$

so dass gilt:

$$\underline{U} \cdot \underline{X} + \underline{L} = \underline{U}_E \cdot \underline{Y} \qquad \text{Gl. (24)}$$

| **Einspannung** | **Gelenkige Lagerung** | **Freier Rand** | **Schwebende Einspannung** |
|---|---|---|---|
| | | | |
| $\underline{Z} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} M_a \\ Q_a \end{bmatrix}$ | $\underline{Z} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} j_a \\ Q_a \end{bmatrix}$ | $\underline{Z} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} w_a \\ j_a \end{bmatrix}$ | $\underline{Z} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} w_a \\ M_a \end{bmatrix}$ |

Tabelle 5. Randvektoren am Trägeranfang und -ende

oder

$$\begin{bmatrix} \underline{U} & -\underline{U}_E \end{bmatrix} \cdot \begin{bmatrix} \underline{X} \\ \underline{Y} \end{bmatrix} = -\underline{L} \qquad \text{Gl. (25)}$$

Aus der Lösung des 4x4-Gleichungssystems (Gl.25) erhält man die Unbekannten $\underline{X}$ und $\underline{Y}$ am Trägeranfang und am Trägerende.

Danach folgt ein zweiter Rechengang zur endgültigen Bestimmung der Zustandsvektoren an den Abschnittsgrenzen beziehungsweise den Knotenpunkten. Hierzu werden die Größen des Zustandsvektors am Trägeranfang durch Einsetzen von $\underline{X}$ in (Gl.13) ermittelt und dieser Vektor durch fortlaufende Anwendung der Übertragungsbeziehungen (Gl.8) für Abschnitte und (Gl.13) für Punkte über den Träger übertragen. Auf diese Weise erhält man die Durchbiegung, den Biegewinkel, das Biegemoment und die Querkraft an allen betrachteten Stellen des Trägers.

### 2.2.1.7 Berechnung von Durchlaufträgern

Bei Durchlaufträgern tritt beim Übertragungsverfahren an jeder Zwischenstütze eine Auflagerkraft als zusätzliche Unbekannte auf.

Bild 52. Übertragung über eine Zwischenstütze

Man führt nun die Berechnung so fort, als würde am Auflager ein neuer, gelenkig gelagerter Träger beginnen., d.h. man führt einen neuen Anfangsvektor mit der Auflagerkraft $Q_r$ und dem Drehwinkel am Auflager als Unbekannte ein. Den links anschließenden Teil des Durchlaufträgers stellt man als Koppelfeder, die am Anfang des neuen Feldes des Durchlaufträgers wirkt, dar. Dazu ist die Übertragungsmatrix in eine Steifigkeitsmatrix umzuformen. Fasst man Kraft- und Verschiebungsgrößen zu Untermatrizen zusammen, so kann geschrieben werden:

$$\begin{bmatrix} \underline{Z}_u \\ \underline{Z}_F \end{bmatrix} = \begin{bmatrix} \underline{Z}_1 \\ \underline{Z}_2 \end{bmatrix} \cdot \underline{U}_0 + \begin{bmatrix} \underline{L}_u \\ \underline{L}_F \end{bmatrix} \qquad \text{Gl. (26)}$$

Die erste Zeile dieses Gleichungssystems lässt sich umformen zu:

$$\underline{U}_0 = \underline{Z}_1^{-1} \cdot \left( \underline{Z}_u - \underline{L}_u \right) \qquad \text{Gl. (27)}$$

Setzt man diese Beziehung in die zweite Zeile des Gleichungssystems ein, so erhält man:

$$\underline{Z}_F = \underline{K}_K \cdot \underline{Z}_u + \underline{L}_K \,, \qquad \text{Gl. (28)}$$

$$\underline{K}_K = \underline{Z}_2 \cdot \underline{Z}_1^{-1} \qquad \text{Gl. (29)}$$

und

$$\underline{L}_K = \underline{L}_F - \underline{Z}_2 \cdot \underline{Z}_1^{-1} \cdot \underline{L}_u \qquad \text{Gl. (30)}$$

$\underline{K}_K$ wird auch Koppelfedermatrix genannt. Es handelt sich um eine 2 x 2 – Steifigkeitsmatrix, welche die Freiheitsgrade der Verschiebungen $u$ und des Drehwinkels $j$ beschreiben. An Zwischenstützen von Durchlaufträgern wird der Anfangsvektor des neuen Feldes zunächst über diese Feder übertragen, um die Steifigkeit des links angeschlossenen Trägerteils zu berücksichtigen.

#### 2.2.1.8 Eingabe

Wie bereits eingangs erläutert, stellt der Engineering Desktop Funktionen und Oberflächen bereit, um statische Berechnungen durchzuführen. Hierzu können unterschiedliche Wege gegangen werden. Wesentlich daran ist, dass bei den komplexen Funktionen die richtige Abfolge eingehalten wird, um die richtigen Ergebnisse zu erzielen.

In Excel wird dem Benutzer in einem Add-In eine Eingabemaske zur Verfügung gestellt. In MathCAD muss der Benutzer hingegen selbst die für ihn notwendigen Funktionen aufrufen, da MathCAD keine Oberflächenprogrammierung zur Verfügung stellt. Abhilfe schafft ein vordefiniertes Arbeitsblatt, das die Funktionen und Funktionsabläufe enthält und damit lediglich für den Einzelfall abgeändert werden muss. Ebenfalls realisiert wurde die Anbindung an das Datenbankprogramm Microsoft ® Access. Dies erfolgte im Zuge einer Diplomarbeit von [12]. Auf eine nähere Beschreibung soll jedoch verzichtet werden.

Das Programm ermöglicht die Berechnung von Trägern nach Theorie I. und II. Ordnung. Die Eingabe erfolgt abschnittsweise von links (Trägeranfang) nach rechts (Trägerende). Dies bedeutet, dass der Benutzer vor der Eingabe des Systems die einzelnen Felder in Abschnitte aufteilen muss. Eine Abschnittsgrenze tritt an Stellen auf, an denen z.B. eine Einzellast angreift oder eine Linienlast Feldes beginnt oder endet. Abschnittsweise kann die Veränderung längs des Trägers für folgende Systemgrößen berücksichtigt werden:

? Biegesteifigkeit E*I

? Schubsteifigkeit $G*A_Q$

? Längskraft $N_{II}$ bei Theorie II. Ordnung

? Bettungsmodul bei elastischer Bettung

Zusätzlich können folgende Größen am System berücksichtigt werden.

- Federn: Weg-, Dreh-, Schub-, Momentenfedern
- Streckenlasten: Gleichlast, Trapezlast und Temperatur
- Punktlasten: Einzellast und Einzelmoment
- Eingeprägte Einzeleinwirkungen: Knickwinkel und Relativverschiebung
- Lagerabsenkungen und –verdrehungen

Als Ergebnis liefert das Programm die Auflagerkräfte sowie die Momente, die Querkräfte und die Verschiebungen über den Trägerverlauf.

### 2.2.2 Programmierung in JAVA

In diesem Kapitel wird die Zusammensetzung des COM–Objekte–Moduls zur Berechnung von allgemeinen Trägern beschrieben. Dieser Programmmodul ist in der Objektbibliothek DLT.Dll enthalten.

#### 2.2.2.1 Allgemeines zur Bibliothek *DLT.Dll*

Die Datenstruktur von der DLT.DLL ist durch, die Entstehungsgeschichte der Komponente geprägt. Am Anfang stand die von Prof. Dr. H. Werkle in JAVA entwickelte Klasse Dlt zur Verfügung. Sie enthielt die Daten und Funktionen zur Berechnung von Durchlaufträgern nach dem Übertragungsmatrizenverfahren. Sie enthielt bereits Klassen für den Zustandsvektor und die erforderlichen Funktionen zur Durchführung des Übertragungsmatrizenverfahrens sowie Klassen zur Matrizenrechnung. Allerdings enthielt sie ein relativ einfaches Datenmodel. Durch die Steigerung der Komplexität der Aufgabenstellung im Rahmen des Projekts entstand ein Bedarf nach weiteren Modulen, die ein vorgeschrittenes Datenmodel verwalten konnten. So entstanden die Klassen SystemAbschnitt und System Punkt um das System darzustellen, und LastenPunkt, LastenAbschnitt, und LagerabsenkungUndVerdrehung, die die Verwaltung unterschiedlicher Lasten im System übernommen haben.

Der eigentliche Rechenkern der Klasse Dlt ist, mit Ausnahme der Einführung der Lastfälle, unverändert geblieben. Die Einführung der Theorie II Ordnung, führte zu dem Austausch der ursprünglichen Klassen Punkt und Abschnitt gegen PunktNeu und AbschnittNeu, die neue Berechnungsregeln für die Koeffizientenmatrix beinhalten.

#### 2.2.2.2 Die Klassen in *DLT.dll*

In *DLT.dll* sind folgende Klassen implementiert:




Bild 53. Implementierte Klassen in DLT.dll

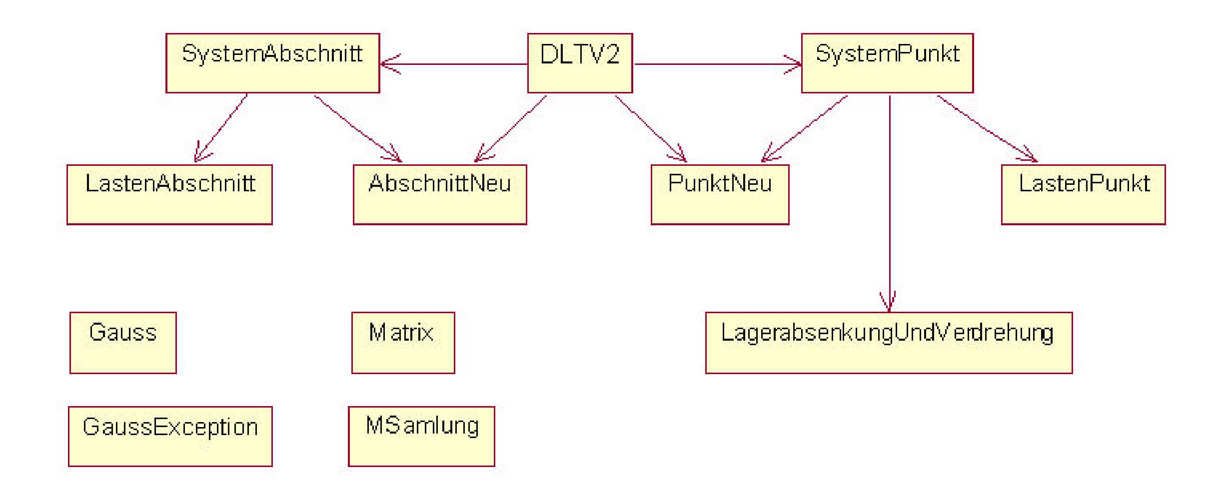



Bild 54. UML-Diagramm

Die COM-Klasse für die Berechnung von Trägern ist die Klasse DLTV2. Sie steht im UML-Diagramm, Bild 2, an zentraler Stelle. Ihre Aufgabe ist die Organisation der Ein- und Ausgabedaten in die Klasse sowie die Durchführung des Rechenablaufs nach dem Übertragungsmatrizenverfahren.

Im folgenden soll nun detailliert auf die einzelnen Klassen eingegangen werden.

### 2.2.2.3 Der Rechenkern – Die Klasse *DLTV2*

Die Diagrammdarstellung der Klassen und Methoden ist in Bild 55 zu sehen. Die wesentlichen Aufgaben der Klasse sind:

- Eingabe des Systems
- Berechnung nach dem Übertragungsmatrizenverfahren
- Ausgabe der Ergebnisse

**COM-Klasse DLTV2**

**Attribute der Klasse *DLTV2***

```
double Auflager[][][] = null;
int anzahlFelder;
int anzahlAbschnitte[];
int anzahlUnterAbschnitte[];
int anzLastfaelle;
AbschnittNeu Absch[][];
PunktNeu     Punk [][];
AbschnittNeu     Absch1 [][];
PunktNeu     Punk1 [][];
char lagerLinks, lagerRechts;
Ergebnisse Erg ;
```

```
SystemAbschnitt listeSA[];
int anzSA;
SystemPunkt listeSP[];
int anzSP;
int theorie2ordnung;
```

**Methoden zur Initialisierung:**

Konstruktoren:

public DltV2()

Die Erste Initialisierung. Die Methode entstand, weil VB keine Parameterübergabe an Konstruktor zuläßt.

```
public void initMatrix(int lL, int lR ,int theorie2ordnungNeu,
                       int anzahlLastFaelle,int anzFelder)
```

Parameter:

```
lL                      // Linkes Auflager – Kennwert 1, 2, 3 oder 4
lR                      // Rechtes Auflager – Kennwert 1, 2, 3 oder 4
theorie2ordnungNeu      // 1 falls Theorie II Ordnung, sonst Null.
anzahlLastFaelle        // Anzahl der Lastfälle
anzFelder               // Anzahl der Felder
```

Für die Lagerbedingungen gelten folgende Kennwerte:

1 gelenkiges Auflager,
2 eingespannt,
3 freier Rand,
4 schwebend eingespannt.

Vorbereitung für die Berechnung:

public void init()

init ist direkt vor dem Aufruf vom *berechnen()* aufrufen.

**Methoden zur Systemeingabe:**

Eingabe der System Abschnitte

```
public void setSystemAbschnitt(int feldNr, int abschnittNr, double laenge, int anzahlDerUnterteilungen,
double E, double schubModul, double I, double schubFlaeche,
double BettModul, double N)
```

Eingabe der System Punkte

```
public void setSystemPunkt(int feldNr, int abschnittNummer, int position, double vertikalFeder ,
double drehFeder, double schubFeder, double momentFeder)
```

Eingabe der Abschnittslasten

```
public void setLastenAbschnitt(int feldNr, int abschnittNr, int lastfallNr,double qLinks,
                               double qRechts, double deltaT, double alphaT,double h)
```

Eingabe der Punktlasten

public void setLastenPunkt(int feldNr, int abschnittNr, int lastfallNr, int position, double F, double M, double deltaW, double deltaPhi)




Bild 55. Die Bibliothek DLTV2

public void setLagerabsenkungVerdrehung(int feldNr,int position, it lastfallNr,double deltaW, double deltaPhi)

Eingabe der Lagerabsenkungen bzw. Verdrehungen.

**Methode zur Durchführung der Berechnung:**

public void berechnen()

Diese Methode beinhaltet den allgemeinen Rechenablauf des Übertragungsmatrizen-verfahrens. Grundlage sind Objekte vom Typ Zustandsvektor. Sie werden in der Klasse Zust definiert. Für die Klasse des Zustandsvektors stehen folgende Funktionen zur Verfügung:

- Übertragung über einen Abschnitt nach (3): Zust uebertrage (Abschnitt Abaktuell)
- Übertragung über einen Punkt nach (5): Zust uebertrage (Punkt Puaktuell)
- Bestimme Koppelfeder nach (22, 23): PunktNeu bestimme Koppelfeder()
- Aufstellen des Anfangsvektors nach Tabelle 5: double [][] bestimmeAnfangswerte()

In der Funktion *berechnen()* der Klasse *DLTV2* werden zunächst Objekte vom Typ *Zust* für den Trägeranfang, das Trägerende, die Zwischenauflager und die jeweils aktuelle Position im Träger erzeugt. Sie stellen Speicherplatz für die Anfangsvektoren, Koppelmatrizen, Abschnittsmatrizen, Abschnittslastvektoren, Punktmatrizen sowie Punktlastvektoren bereit. Die Berechnung wird gleichzeitig für alle Lastfälle durchgeführt, d.h. der Zustandsvektor $\underline{Z}_r$ in (3) wird zu einer Matrix erweitert, in der jede Spalte einem Lastfall entspricht.

Kern der Berechnung nach dem Übertragungsmatrizenverfahren sind zwei Schleifen, und zwar eine Schleife über alle Felder und innerhalb dieser Schleife, eine weitere Schleife über alle Abschnitte. In der Schleife über die Abschnitte wird der Zustandsvektor über alle Abschnitte und Punkte des jeweiligen Feldes übertragen und am Feldende wird die Koppelfedermatrix aufgestellt. Mit dieser wird der Anfangsvektor des nächsten Feldes multipliziert und die Übertragung über das nächste Feld durchgeführt. Am Ende des Trägers wird mit Hilfe des Zustandsvektors für das Trägerende das Gleichungssystem nach (18) aufgestellt und gelöst. Damit steht der Anfangsvektor des letzten Feldes zur Verfügung.

In einem zweiten Berechnungsgang wird die Übertragung des Zustandsvektors, beginnend beim letzten Feld, nochmals durchgeführt. Die dabei erhaltenen Ergebnisse werden in einem Objekt der Klasse Ergebnisse abgespeichert.

Für die Matrizenrechnung werden Objekte der Klasse Matrix und deren Funktionen verwendet. Die Lösung des Gleichungssystems erfolgt mit der Klasse Gauss. Zur Übertragung des Zustandsvektors werden die zuvor bei der Initialisierung (Funktion *init*) mit den Klassen *Abschnittneu* und *Punktneu* erzeugten Felder für die Abschnitts- und Punktmatrizen herangezogen.

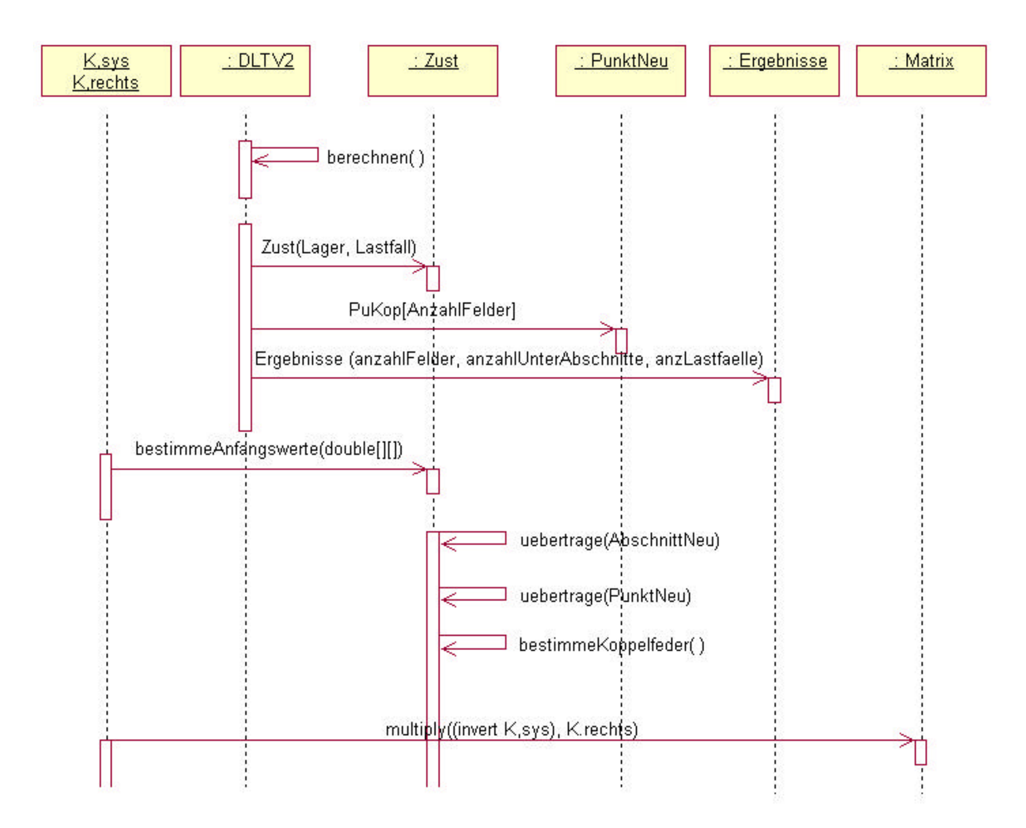



Bild 56. Sequenz-Diagramm der Methode berechnen()

**Methoden zur Ergebnisübergabe:**

Gibt das Moment an der angegebenen Position zurück:
public double getM(int lastFahlNr,int feldNr,int unterAbschnitt, int position)

Gibt Querkraft an der angegebenen Position zurück:
public double getQ(int lastFahlNr,int feldNr,int unterAbschnitt, int position)

Gibt Verdrehung an der angegebenen Position zurück:
public double getW(int lastFahlNr,int feldNr,int unterAbschnitt, int position)

Gibt das maximale Moment zurück:
public double getMmax(int feldNr,int unterAbschnitt, int position)

Gibt die maximale Querkraft zurück:
public double getQmax(int feldNr,int unterAbschnitt, int position)

Gibt die maximale Verdrehung zurück:
public double getWmax(int feldNr,int unterAbschnitt, int position)

Gibt das minimale Moment zurück:
public double getMmin(int feldNr,int unterAbschnitt, int position)

Gibt die minimale Querkraft zurück:
public double getQmin(int feldNr,int unterAbschnitt, int position)

Gibt die minimale Verdrehung zurück:
`public double getWmin(int feldNr,int unterAbschnitt, int position)`

Gibt maximales Moment an einem Auflager aus:
`public double getAuflagerMaxM(int AuflagerNr)`

Gibt minimales Moment an einem Auflager aus:
`public double getAuflagerMinM(int AuflagerNr)`

Gibt maximale Querkraft an einem Auflager aus:
`public double getAuflagerMaxQ(int AuflagerNr)`

Gibt minimale Querkraft an einem Auflager aus:
`public double getAuflagerMinQ(int AuflagerNr)`

Gibt das Moment an einem Auflager bei einem bestimmten Lastfall:
`public double getAuflagerM(int AuflagerNr, int LastFall)`

Gibt die Querkraft an einem Auflager bei einem bestimmten Lastfall:
`public double getAuflagerQ(int AuflagerNr, int LastFall)`

Gibt das maximale Moment in einem Feld:
`public double MaximalesFeldMoment(int feldNr)`

Gibt die x-Koordinate des maximalen Momentes in einem Feld:
`public double MaximalesFeldMomentX(int feldNr)`

## Interne Klasse *Zust* der Zustandsvektoren

Die Klasse *Zust* beschreibt Zustandsvektoren. Sie enthält Methoden zur Übertragung über Abschnitte und Punkte sowie zur Erstellung von Koppelfedermatrizen nach dem letzten Abschnitt eines Feldes.

**Attribute der Klasse *Zust*:**

```
double Zm[][] =new double [4][2];
double Zl[][];// = new double [][];
int lastFaelle;
```

**Methoden:**

Default Konstruktor:
`public Zust()`

Konstruktor: Zustandsvektor am Anfang und Ende festlegen:
`public Zust(char lagerbedingung,int anzLastfaelle)`

Übertragen des Zustandsvektors:
```
public Zust uebertrage(AbschnittNeu AbAktuell)//Übertragung über Abschnitte
public Zust uebertrage(PunktNeu PuAktuell )   //Übertragung über Punkte
```

Bestimmung der Koppelfeder:
```
public PunktNeu bestimmeKoppelfeder()
```

Gibt eine Kopie der Instanz bei der sie aufgerufen wird:
```
public Zust kopiere()
```

Bestimmt die Anfangswerte:
```
public double [][] bestimmeAnfangswerte(double [][] xAnfang)
```

**Die interne Klasse *Ergebnisse***

**Attribute:**
```
        int nr
        double ergAnPunkt [][][][]
```

**Methoden:**

Konstruktor:
```
public Ergebnisse(int anzahlFelder, int anzahlAbschnitte[],
                        int anzLastfaelle)
```

Übernehmen des Zustandsvektors und der Anfangswerte sowie
Berechnen und Speichern der Ergebnisse:
```
public void uebernehmen(int nrFeld, Zust Z, double [][] Loes)
```

### 2.2.2.4 Besonderheiten der Klasse *DLT V2*

Als einzige Klasse besitzt die Klasse *DltV2* keinen Entry Point, sondern läuft nach einer starr vordefinierten Struktur ab. Dies liegt daran, dass die Methoden der Klasse nicht in einer beliebigen Reihenfolge aufgeführt werden dürfen.

Die Reihenfolge der Aufrufe:

initMatrix(),
Eingabe der Systemdaten(setSystemPunkt, setSystemAbschnitt),
Eingabe der Lasten(setLastenPunkt setLastenAbschnitt),
init(),
berechnen(),
Ausgabe der Daten.

Sämtliche Zähler für Felder, Abschnitte, Unterabschnitte, Lasten, etc. beginnen bei Eins. Für die Definition der Positionen der Punkte[2] sind numerische Werte definiert.

[2] 1 für den Anfang des Abschnittes, 2 für das Ende.

### 2.2.2.5 Klassen für Abschnitte

Diese Klassen erzeugen die erforderlichen Parameter für die einzelnen System- und Unterabschnitte. Die Klasse *AbschnittNeu* stellt die Abschnittsmatrizen und die Abschnittslasten bereit. Diese werden für den geraden Stab mit konstantem Querschnitt nach den Theorien I. und II. Ordnung aufgestellt. Es gehen der Einfluss einer elastische Bettung sowie die Berücksichtigung der Schubsteifigkeit $G \cdot A_Q$ in die Koeffizienten ein. Die erforderlichen Beiwerte $a_j$ und $b_j$ werden mittels der Klasse *MSammlung* erzeugt.

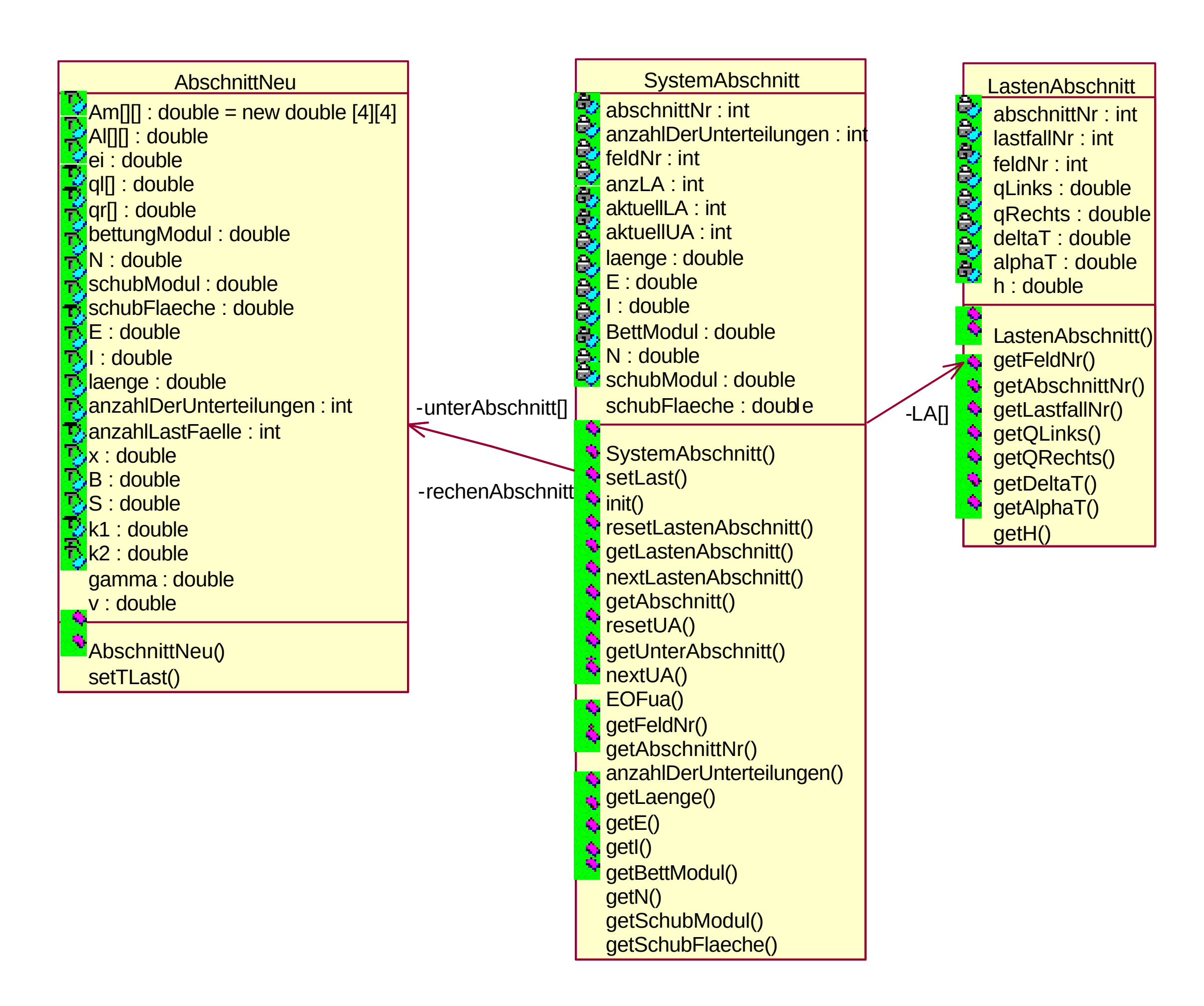



Bild 57. UML-Darstellung der Klassen für die Abschnitte

Nachdem die Matrizen bereitgestellt wurden, werden sie in der Klasse *SystemAbschnitt* mit den Eingabedaten gefüllt, die in DLTV2 eingelesen und weitergegeben werden. Für die Klasse *LastenAbschnitt* gilt dies in gleicher Weise für die Abschnittslasten.

#### Die Klasse *AbschnittNeu*

Die Klasse *AbschnittNeu* dient der Aufstellung der Abschnittsmatrizen und der Abschnittslasten. Die Klasse enthält die Koeffizientenmatrix für den geraden Stab mit konstantem Querschnitt nach den Theorien I. und II. Ordnung mit dem Einfluss einer elastische Bettung sowie der Berücksichtigung der Schubsteifigkeit $G \cdot A_Q$ nach [4]

**Attribute:**

```
double Am[][] = new double [4][4];
double Al[][];
double ei, ql[], qr[], bettungModul, N,schubModul, schubFlaeche;
double E, I, laenge;
int anzahlDerUnterteilungen, anzahlLastFaelle;
double x, B, S, k1, k2, gamma, v;
```

**Methoden:**

Konstruktor:

```
public AbschnittNeu( double myql[], double myqr[], double mybettungModul,
                     double myN, double myschubModul, double myschubFlaeche,
                     double myE, double myI, double mylaenge,
                     int myanzahlDerUnterteilungen, int myanzahlLastFaelle)
```

Es wird an dieser Stelle die 4 x 4 – Koeffizientenmatrix erstellt.

Außerdem ist eine Temperaturlast für den Abschnitt definiert:

```
public void setTLast(double deltaT, double alphaT, double h, int lastFall)
```

darin sind:

deltaT ist die Differenz zwischen Querschnittsober- und Unterkante,
alfaT der spezifische Temperaturkoeffizient des Baustoffes,
h Querschnittshöhe, lastFall Lastfallnummer, in der die Temperatur berücksichtigt werden soll.

Wird in einem Abschnitt innerhalb eines Lastfalles eine Temperaturlast definiert, werden die eventuell vorhandenen Abschnittslasten für diesen Abschnitt innerhalb dieses Lastfalls nicht berücksichtigt. Daher muss dieser Lastfall separat berechnet werden.

### Die Klasse *SystemAbschnitt*

**Attribute:**

```
private int abschnittNr, anzahlDerUnterteilungen, feldNr;
private AbschnittNeu rechenAbschnitt;
private AbschnittNeu unterAbschnitt[];
private LastenAbschnitt LA[];
private int anzLA;
private int aktuellLA, aktuellUA;
private double laenge, E, I, BettModul, N, schubModul, schubFlaeche;
```

**Methoden:**

Konstruktor

```
public SystemAbschnitt(int feldNrNeu, int abschnittNrNeu, double laengeNeu,
                       int anzahlDerUnterteilungenNeu, double ENeu,
                       double INeu, double BettModulNeu, double NNeu,double
                       schubModulNeu, double schubFlaecheNeu)
```

Eingabe einer Abschnittslast:
public int setLast (LastenAbschnitt lAbschnitt)

Initialisierung:
public void init(int anzahlLastFaelle)

Methoden zur Ausgabe der Attribute:

```
public void resetLastenAbschnitt()
public LastenAbschnitt getLastenAbschnitt()
public void nextLastenAbschnitt()
public AbschnittNeu getAbschnitt()
public void resetUA()
public AbschnittNeu getUnterAbschnitt()
public void nextUA()
public int EOFua()
public int getFeldNr()
public int getAbschnittNr()
public int anzahlDerUnterteilungen()
public double getLaenge()
public double getE()
public double getI()
public double getBettModul()
public double getN()
public double getSchubModul()
public double getSchubFlaeche()
```

## Die Klasse *LastenAbschnitt*

**Attribute:**

```
private int abschnittNr, lastfallNr,feldNr;
private double qLinks, qRechts, deltaT;
private double alphaT, h;
```

**Methoden:**

Konstruktor

```
public LastenAbschnitt(int feldNrNeu,int abschnittNrNeu,int lastfallNrNeu,
                       double qLinksNeu,double qRechtsNeu,double deltaTNeu,
                       double alphaTNeu, double hNeu)
```

Methoden zur Ausgabe von Attributen

```
public int getFeldNr()
public int getAbschnittNr()
public int getLastfallNr()
public double getQLinks()
public double getQRechts()
public double getDeltaT()
public double getAlphaT()
```

```
public double getH()
```

### 2.2.2.6 Klassen für Punkte

Die Berechnung der Knotenpunkte ist ebenfalls wie die Abschnittsberechnung in mehrere Klassen aufgeteilt. Ein Grundgerüst für die Matrizen erzeugt die Klasse *PunktNeu*, welche die Punktmatrix und die Koppelmatrix liefert. Die Klasse *LastenPunkt* steht für die Aufbringung von Punktlasten. Die Klasse *LagerabsenkungUndVerdrehung* stellt eingeprägte Lagerverformungen dar. Alle zusammen werden durch die Klasse *SystemPunkt* angesteuert, die aus *DLTV2* ihre Werte bezieht. Bild 58 zeigt die UML-Klassendiagramm für die Berechnung an den Punkten.

**Klasse *SystemPunkt***

**Attribute:**

```
private LastenPunkt LP[];
private LagerabsenkungUndVerdrehung LAuV[];
private int anzLP,anzLAuV;
private int aktuellLP,aktuellLAuV;
private int abschnittNr,feldNr,position;
private PunktNeu punktMatrix;
private double vertikalFeder, drehFeder,schubFeder, omentFeder;
```

**Methoden:**

Konstruktor

```
public SystemPunkt(int feldNrNeu, int abschnittNrNeu, int positionNeu,
                   double vertikalFederNeu ,double drehFederNeu,
                   double schubFederNeu, double momentFederNeu)
```

Übergibt die eingelesenen Variablen an die Klasseninternen Parameter

Eingabe einer Punktlast.

```
public int setLast(LastenPunkt lPunkt)
```

Eingabe einer Lagerabsenkung oder Verdrehung

```
public int setLast(LagerabsenkungUndVerdrehung lAuV)
```

Initialisierung

```
public void init(int anzLastFaelle)
```

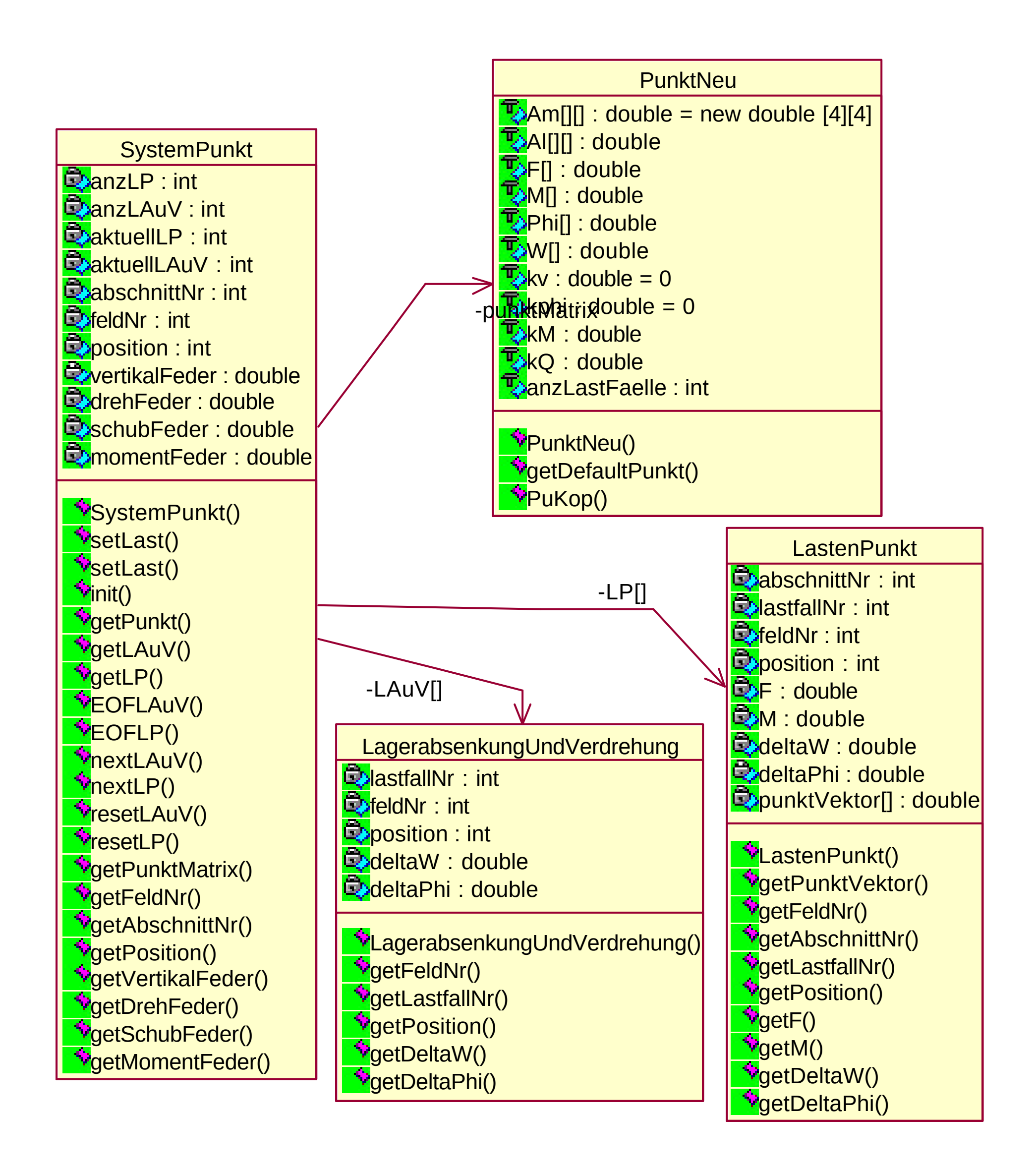



Bild 58. UML-Darstellung für die Punkte

## Ausgabe von Attributen

public PunktNeu getPunkt()
public LagerabsenkungUndVerdrehung getLAuV()
public LastenPunkt getLP()
public int EOFLAuV()
public int EOFLP()
public void nextLAuV()
public void nextLP()
public void resetLAuV()
public void resetLP()
public PunktNeu getPunktMatrix()
public int getFeldNr()
public int getAbschnittNr()
public int getPosition()
public double getVertikalFeder()

```
public double getDrehFeder()
public double getSchubFeder()
public double getMomentFeder()
```

**Klasse *PunktNeu***

Die Klasse *PunktNeu* dient der Aufstellung der Punktmatrizen. Sie liefert eine 4x4 – Matrix

**Attribute:**

```
double Am[][] = new double [4][4];
double Al[][];
double F[], M[], Phi[],W[];        // Verschiebungsvektor rechts des Punktes
double kv=0, kphi=0;               // Federsteifigkeiten
double kM, kQ;                     //
int anzLastFaelle;                 // Anzahl der vorhanden Lastfälle
```

**Methoden:**

Konstruktor

```
public PunktNeu(double myF[], double myM[], double myPhi[], double myW[],
         double myCv, double myCphi, double myCM, double myCQ,int
         myAnzLastFaelle)
```

Gibt einen leeren Punkt aus:

```
public PunktNeu getDefaultPunkt()
```

**Klasse *LastenPunkt***

**Attribute:**

```
private int abschnittNr, lastfallNr, feldNr, position;
private double F, M, deltaW, deltaPhi;
private double punktVektor[];
```

**Methoden:**

Konstruktor

```
public LastenPunkt(int feldNrNeu, int abschnittNrNeu, int lastfallNrNeu,
            int positionNeu, double FNeu, double MNeu,
            double deltaWNeu, double deltaPhiNeu)
```

Methoden für die Ausgabe der Attribute:

```
public double[] getPunktVektor()
public int getFeldNr()
public int getAbschnittNr()
public int getLastfallNr()
public int getPosition()
public double getF()
public double getM()
public double getDeltaW()
public double getDeltaPhi()
```

### 2.2.2.7 Weitere Hilfsklassen

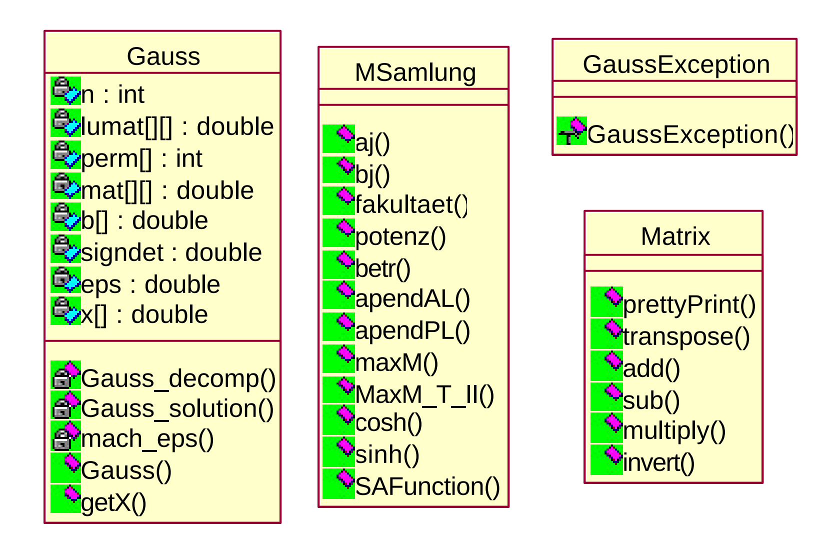



Bild 59. UML-Klassendiagramm weiterer Hilfsklassen

Die Klasse MSammlung beinhaltet sämtliche Methoden, auf die von den unterschiedlichen Klassen unabhängig zugegriffen wird. Es sind die Beiwerte nach Rubin enthalten sowie Methoden, die komplizierte mathematische Java–Ausdrücke ersetzen.

Diese folgenden drei Klassen wurden vom Autor lediglich implementiert. Sie stammen aus dem Internet [3] aus dem Programm „SimpleFE“. Es wurden daran keine Änderungen vorgenommen. Sie werden daher an dieser Stelle zusammen nur kurz vorgestellt.

public class Matrix
public class Gauss
public class GaussException extends RuntimeException.

**Klasse *Matrix***

Klasse von Funktionen zur Matrix-Rechnung mit zweidimensionalen Feldern.

public static void prettyPrint(double matrix[][]) // Liefert einen Standardausdruck zu Testzwecken

public static double[][] transpose(double a[][]) // liefert die transponierte Matrix von a

public static double[][] add(double a[][], double b[][]) // Addiert die Matrizen a und b

public static double[][] sub(double a[][], double b[][]) // Subtrahiert die Matrix b von a

public static double[][] multiply(double a[][], double b[][]) // Multipliziert die Matrizen a und b

public static double[][] invert(double a[][]) // liefert die inverse Matrix von a

[3] http://www.ee.oulu.fi/~timor/javaa

Die Matrixklasse stellt die Grundrechenfunktionen von Matrizen bereit. Um diese ausführen zu können werden Hilfsklassen benötigt, um Gleichungssysteme zu lösen.

**Die Klasse *Gauss***

```
private int Gauss_decomp()                    // Regeln für Fehlerermittlung (z.B. Singularität)
private int Gauss_solution()
private double mach_eps()
public Gauss(double[][] M, double[] b)        // Löst das LGS 'M*x=b' auf
public double[] getX()                        // Liefert den Lösungsvektor 'x' zurück
```

Hier wird ein Gleichungslöser nach dem Gauss'schen Algorithmus bereitgestellt. Dabei werden auftretende Fehler erkannt und Ausgeworfen (THROW EXEPTION)

**Die Klasse *GaussException***

Damit die Fehler Abgefangen werden können existiert die Klasse `GaussException.java:`

```
public class GaussException extends RuntimeException   // liefert Fehlermeldungen als String zurück wie
                                                        // in der RuntimeException enthalten
```

### 2.2.2.8 Eingabe

Wie bereits eingangs erläutert, stellt der Engineering Desktop Funktionen und Oberflächen bereit, um eine Berechnung ausführen. Hierzu können unterschiedliche Wege gegangen werden. Zum Einen können die Funktionen direkt aufgerufen werden, zum Anderen kann ein existierendes Arbeitsblatt oder Excel–Sheet verwendet werden. Wesentlich daran ist, dass bei den komplexen Funktionen die richtige Abfolge eingehalten wird, um die richtigen Ergebnisse zu erzielen. In Excel wird daher dem Benutzer eine Eingabemaske präsentiert, damit dieser sich nicht die relevanten Komponenten zusammensuchen muss. In Mathcad muss der User dahingegen selbst die für ihn notwendigen Funktionen aufrufen, da Mathcad keine Oberflächenprogrammierung zur Verfügung stellt. Abhilfe schafft ein vordefiniertes Arbeitsblatt, das die Funktionen und Funktionsabläufe enthält und damit lediglich für den Einzelfall abgeändert werden muss.

Das Programm ermöglicht die Berechnung von Durchlaufträgern nach Theorie I. und II. Ordnung. Die Eingabe erfolgt abschnittsweise von links (Trägeranfang) nach rechts (Trägerende). Abschnittsweise im Sinne des Durchlaufträgerprogramms bedeutet, dass der Benutzer vor der Eingabe des Systems die einzeln Felder in Abschnitte aufteilen muss. Eine Abschnittsgrenze liegt z.B. vor, wenn eine Einzellast innerhalb eines Feldes angreift oder eine Linienlast innerhalb eines Feldes beginnt oder endet. können Veränderungen folgender Systemgrößen längs des Trägers können abschnittsweise berücksichtigt werden:

- Biegesteifigkeit $E \cdot I$
- Schubsteifigkeit $G \cdot A_Q$
- Längskraft $N_{II}$ bei Theorie II. Ordnung
- Bettungsmodul bei elastischer Bettung

Zusätzlich können folgende Größen am System berücksichtigt werden.

- Federn: Verschiebungs- und Drehfedern, innere Querkraft- und Momentenfedern
- Streckenlasten: Gleichlast und Trapezlast
- Temperaturlasten
- Punktlasten: Einzellast und Einzelmoment
- Eingeprägte Relativverschiebungen und Knickwinkel zur Bestimmung von Einflusslinien
- Lagerabsenkungen und –verdrehungen

Als Ergebnis liefert das Programm die Auflagerkräfte sowie die Momente, die Querkräfte und die Verschiebungen über den Trägerverlauf.

## 2.3 Ebenes Fachwerk

### 2.3.1 Theoretische Grundlagen

Die Berechnung des Fachwerks erfolgt auf der Grundlage der Methode der Finiten Elemente. Der Berechnungsablauf hierzu ist in [5] beschrieben. Es wird das in Bild 1 dargestellte globale Koordinatensystem zugrunde gelegt.

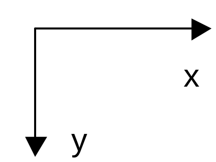



Bild 60. Globales Koordinatensystem

#### 2.3.1.1 Elementsteifigkeitsmatrix

Ebene Fachwerke besitzen an jedem Knotenpunkt zwei Freiheitsgrade. Bei einem Fachwerk in der x-y-Ebene sind dies die Verschiebungen $u$ und $v$. Die entsprechenden Stabendkräfte sind $F_x$ und $F_y$, Bild 2.

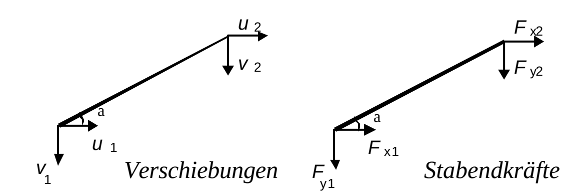



Bild 61. Verschiebungen und Stabendkräfte bei Fachwerkknoten

Die Elementsteifigkeitsmatrix ist damit:

$$\frac{AE}{l}\begin{bmatrix} c^2 & s\cdot c & -c^2 & -s\cdot c \\ s\cdot c & s^2 & -s\cdot c & -s^2 \\ -c^2 & s^2 & c^2 & s\cdot c \\ -s\cdot c & -s^2 & s\cdot c & s^2 \end{bmatrix}\cdot\begin{bmatrix} u_1^{(e)} \\ v_1^{(e)} \\ u_2^{(e)} \\ v_2^{(e)} \end{bmatrix}=\begin{bmatrix} F_{x1}^{(e)} \\ F_{y1}^{(e)} \\ F_{x2}^{(e)} \\ F_{y2}^{(e)} \end{bmatrix} \qquad \text{Gl. (31)}$$

bzw.:

$$K_e * u_e = F_e \qquad \text{Gl. (32)}$$

mit den Parametern:
- $A$: Querschnittsfläche des Stabes
- $E$: Elastizitätsmodul des Stabes
- $l$: Stablänge
- $s$: sin α
- $c$: cos α

#### 2.3.1.2 Systemsteifigkeitsmatrix $\underline{K}_{ges}$

Aus den Steifigkeitsmatrizen wird die Systemsteifigkeitsmatrix, die auf alle Freiheitsgrade des statischen Systems bezogen ist, gebildet. Hierzu werden die Elementssteifigkeitsmatrizen auf die Systemsteifigkeitsmatrix aufaddiert.

#### 2.3.1.3 Kräftevektor $\underline{F}_{ges}^{Last}$

Der Kräftevektor wird aus den an den Knotenpunkten angreifenden äußeren Lasten gebildet..

#### 2.3.1.4 Gleichungssystem

Damit kann nun das Gleichungssystem für die gesamte Struktur gebildet werden. Es gilt:

$$\underline{K}_{ges} \cdot \underline{u}_{ges} = \underline{F}_{ges}^{Last} \qquad \text{Gl. (33)}$$

Dieses Gleichungssystem ist so jedoch nicht lösbar, da es zu viele Unbekannte besitzt. Da allerdings die Verschiebungen der festgehaltenen Freiheitsgrade gleich Null und damit bekannt sind, kann das Gleichungssystem auf ein lösbares System reduziert werden. Hierzu werden die Zeilen und Spalten, die den festgehaltenen Freiheitsgraden entsprechen, eliminiert. Es folgt :

$$\underline{K}_{red} \cdot \underline{u}_{red} = \underline{F}_{red} \qquad \text{Gl. (34)}$$

Durch Lösen des Gleichungungssystems mit dem Gauss'schen Verfahren erhält man den Vektor $\underline{u}_{red}$ der Verschiebungen..

#### 2.3.1.5 Auflagerkräfte

Um die Auflagerkräfte zu erhalten wird wieder die Gesamtsteifigkeitsmatrix $\underline{K}_{ges}$ herangezogen. Diesmal wird diese dahingehend modifiziert, dass

- die Spalten der Koeffizienten an denen sich Festhaltungen befinden
- sowie die Zeilen die *keine* Festhaltungen besitzen

eliminiert werden. Es entsteht eine Rechteckmatrix $\underline{K}_{red,Auflager}$ .

Man erhält die Auflagerkräfte durch Multiplikation mit dem Verschiebungsvektor.

$$\underline{F}_{Auflager} = \underline{K}_{red,Auflager} \cdot \underline{u}_{red} \qquad \text{Gl. (35)}$$

#### 2.3.1.6 Schnittgrößen am Einzelstab

Aus der bisherigen Berechnung sind die Verschiebungen an den Knotenpunkten vollständig bekannt, da an den Festhaltungen die Verschiebungen mit Null angesetzt werden können. Die Schnittgrößen ermitteln sich damit für jeden Einzelstab nach folgender Gleichung.

$$N = \underline{S}_e \cdot \underline{u}_e \qquad \text{Gl. (36)}$$

$$\underline{S}_e = \frac{AE}{l} \cdot \left[-\cos a \quad -\sin a \quad \cos a \quad \sin a\right] \qquad \text{Gl. (37)}$$

### 2.3.2 Umsetzung in JAVA

Die Adaption des JAVA-Codes zur statischen Berechnung von Fachwerken und deren Anbindung an den Engineering Desktop erfolgte im Rahmen eines studentischen Projektes und wurde auf der Grundlage einer vorgegeben JAVA-Software entwickelt [6]..

#### 2.3.2.1 Klassenstruktur

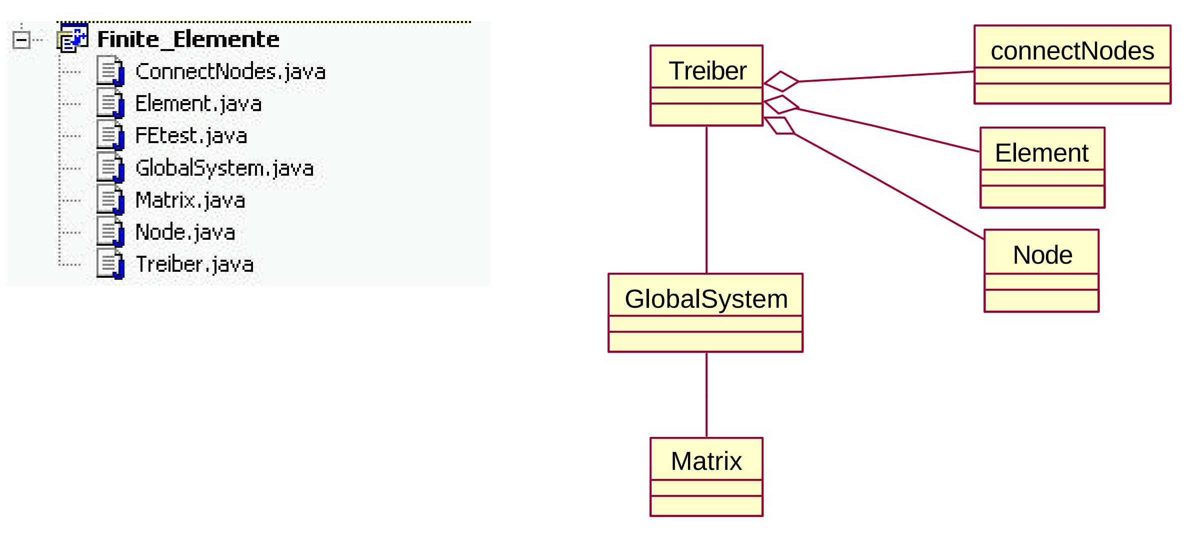



Bild 62. UML-Diagramm *Fachwerk*

Die verwendeten Klassen und deren interner Zusammenhang sind in Bild 1 dargestellt: Es gibt zwei Arten von Klassen: Eine davon stellt die Objektschnittstelle für die Anwenderprogramme bereit, und die andere ist zuständig für das Lösen des statischen Problems.

Bei der Erstellung wurde Wert darauf gelegt, dieses Objekt so zu strukturieren, dass es jederzeit erweitert werden kann. Es sind Schnittstellen vorhanden, die es dem Anwenderprogramm erlauben, sämtliche Daten der Objekt-DLL zu übergeben und jederzeit auf diese Daten wieder zuzugreifen. Damit kann aber auch die DLL um Funktionalitäten erweitert werden. Weiteren Funktions-Klassen stehen sämtliche Daten intern bereit.

#### 2.3.2.2 Objekt-Schnittstellen

Die für den Anwender der DLL wichtige Klasse heißt *Treiber*. Sie ist ein Adapter, der sämtliche Daten zwischenspeichert und die Übergabe an die Berechnungsklasse *GlobalSystem* übernimmt. Das hat den Vorteil, dass weitere Algorithmen leicht integriert werden können.

| Treiber |
|---|
| elemForce[] : double = new double [300]<br>force[] : double<br>boundary[] : int |
| Treiber()<br>addNode(x : double, y : double) : int<br>addElem(node1 : int, node2 : int, myA : double, myE : double) : int<br>addForce(pos : int, x : double, y : double) : int<br>addBound(num : int, x : int, y : int) : int<br>getNumberNodes() : int<br>getNodeX(pos : int) : double<br>getNodeY(pos : int) : double<br>getNumElem() : int<br>solve() : double<br>getResult(pos : int) : double<br>getForce(pos : int) : double<br>getForce_A(pos : int) : double |

Bild 63. UML Klassendiagramm der Klasse *Treiber*

**Methoden zur Eingabe**

public int addNode(double x,double y)
Knotenkoordinaten in [m]. Nummerierung erfolgt intern nach der Reihenfolge der Eingabe, beginnend mit 1.
x, y: Koordinatenwerte in [m]
Return: Nummer des Knotens (1,2,...,n)

public int addElem(int node1, int node2, double myA, double myE)
Eingabe der Stäbe. Welcher Knoten ist mit welchem verbunden? Die Nummerierung erfolgt wieder intern nach der Reihenfolge der Eingabe, beginnend mit 1.
node1, node2: Knotennummer
myA: Stab Fläche
myE: Stab E-Modul
Return: Nummer des Elements (1,2,..,n)

public int addForce(int pos, double x, double y)
Eingabe der Stabkräfte pro Knoten in [kN]. Default-Wert ist NULL.
pos: Nummer des Knotens (1-...)
x, y: Kraftvektoren in x/y

public int addBound(int num, int x, int y)
Eingabe der Auflagerfesthaltungen. Initialisiert wird jeder Knoten mit 1 (-> nicht gelagert).
num: Knotennummer (1-...)
x, y = -1 Festhaltung in x- bzw. y-Richtung (1: nicht gelagert; -1 gelagert)

**Methode zur statischen Berechnung:**

double solve()
Führt die statische Berechnung des Fachwerks durch. Danach stehen die Knotenverschiebungen und Stabschnittgrößen bereit und können mit Hilfe der Output-Schnittstellen ausgelesen werden.

**Methoden zur Ausgabe:**

public double getResult(int pos)
Es können alle Knotenverschiebungen in [mm] ausgelesen werden. Die Rückgabe erfolgt im Wechsel von x- und y- Koordinaten. Somit muss der Aufruf der Methode doppelt so hoch sein wie die Anzahl an Knoten.

public double getForce(int pos)
Liefert die Kräfte in den Stäben in [kN] zurück.
pos: Nummer des Stabs
Return: Stabkraft in [kN]

public double getForce_A(int pos)
Liefert die Spannung im Stab, d.h. die Kraft des Stabes geteilt durch den Wert A der Querschnittsfläche des Fachwerkstabes.
pos: Nummer des Stabs
Return: Stabkraft in [kN] / A

Als weitere Schnittstellen stehen folgende Methoden zur Verfügung:

public int getNumberNodes()
Liefert die Anzahl an Knoten.

public double getNodeX(int pos)
Liefert die x-Koordinate des Knotens pos.

public double getNodeY(int pos)
Liefert die y-Koordinate des Knotens pos.

public int getNumElem()
Liefert die Anzahl an Stäben (Elemente).

### 2.3.2.3 Interne Schnittstellen

*GlobalSystem* ist die Klasse, die die Berechnung des Tragwerks übernimmt. Derzeit übernimmt die Methode *Treiber.solve()* die Übergabe und stellt die Ergebnisse in einer eigenen Vektorklasse bereit. Die im Folgenden genannten Methoden werden intern von dem Adapter aufgerufen.
Notwendig ist die Klasse für die Aufstellung der Gesamtsteifigkeitsmatrix. Es werden Methoden bereitgestellt, um den Lastvektor aufzustellen und die Auflagerbedingungen zu berücksichtigen. Mit *solveThisSystem* wird der Lösungsvektor des Systems erhalten. Ebenso enthalten sind die Algorithmen für die Bestimmung der Schnittgrößen.

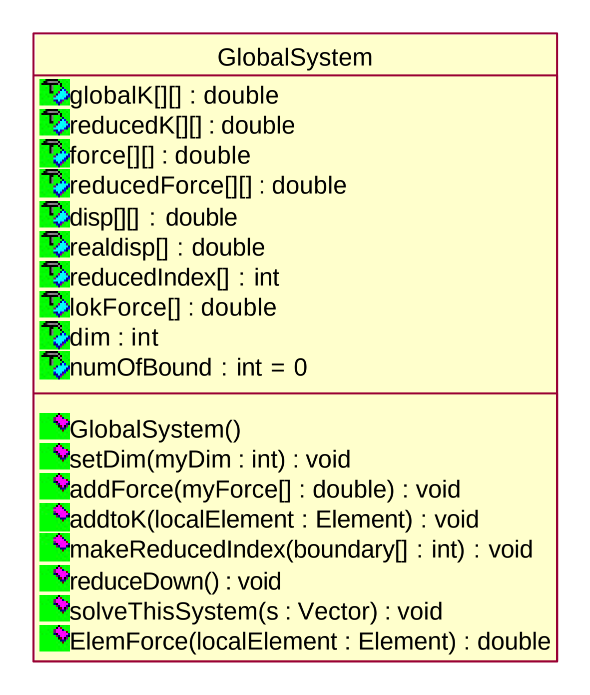



Bild 64. UML-Darstellung der Klasse GlobalSystem

```
global.setDim(getNumberNodes()*2)
global.addForce(force)
global.makeReducedIndex(boundary)
global.reduceDown()
global.solveThisSystem(solves)
```

Dazu wurde eine im Internet veröffentliche existierende Java-Klasse verwendet, die entsprechende Features mitbringt um Matrizen zu addieren, zu subtrahieren, zu invertieren und zu multiplizieren. [7]. Diese Klasse wurde bereits beim Programm für den allgemeinen Träger verwendet.

| Matrix |
|---|
| |
| Matrix()<br>prettyPrint(matrix[][] : double) : void<br>print(matrix[][] : double) : void<br>transpose(matrix[][] : double, inPlace : boolean) : double[][]<br>add(arg1[][] : double, arg2[][] : double, result[][] : double) : double[][]<br>add(arg1[][] : double, arg2[][] : double) : double[][]<br>subtract(arg1[][] : double, arg2[][] : double, result[][] : double) : double[][]<br>subtract(arg1[][] : double, arg2[][] : double) : double[][]<br>multiply(arg1 : double, arg2[][] : double, result[][] : double) : double[][]<br>multiply(arg1 : double, arg2[][] : double) : double[][]<br>multiply(arg1[][] : double, arg2[][] : double, result[][] : double) : double[][]<br>multiply(arg1[][] : double, arg2[][] : double) : double[][]<br>dotMultiply(arg1[][] : double, arg2[][] : double) : double<br>luDecompose(matrix[][] : double, indx[] : int, parity : boolean) : double[][]<br>luBackSubstitute(a : double[][], b : double[][], indx[] : int) : double[][]<br>solve(a[][] : double, b[][] : double) : double[][]<br>invert(arg[][] : double) : double[][]<br>main(args[] : String) : void |

Bild 65. UML-Darstellung der Klasse *Matrix*

*Element* stellt die Berechnung der Elementsteifigkeitsmatrix bereit. Diese ist nach Gleichung (24) aufgebaut. Länge und Winkel ergeben sich aus den Knotenpunkten für Stabanfang und –ende.

| Element |
|---|
| k[][] : double = new double [4][4]<br>x1 : double<br>y1 : double<br>x2 : double<br>y2 : double<br>AE : double<br>L : double<br>c : double<br>s : double<br>node1 : int<br>node2 : int<br>Stres[] : double = new double [4] |
| Element(myNode1 : int, myNode2 : int, myX1 : double, myY1 : double, myX2 : double, myY2 : double, myAE : double) |

Bild 66. UML-Darstellung der Klasse *Element*

*Node* liest die Knotenpunkte der Treiberklasse ein und verarbeitet deren Information. Die Knoten werden strukturiert und den weiteren Klassen (z.B. *Element*) zur Verfügung gestellt. Außerdem bereitet diese Klasse die Kräfteeingabe und die Festhaltungen auf.

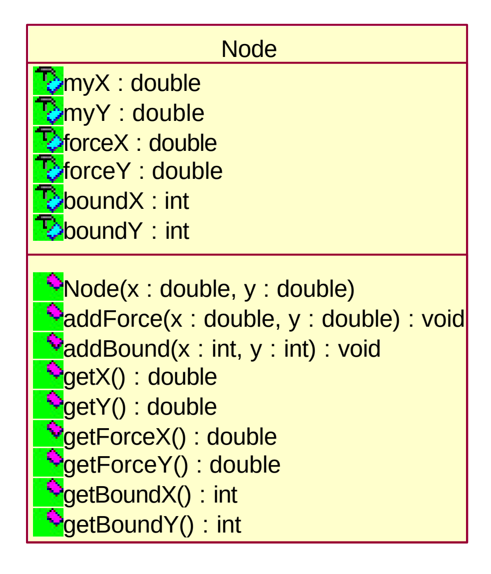



Bild 67. UML-Darstellung der Klasse Element

*ConnectNodes* dient der Parameterübergabe der Knoten- und Elementdaten.

Der gesamte Berechnungsablauf ist in *Treiber* enthalten. Die einzelnen Schritte sind:

- STATION 1 Übernahme der Eingabe für die Knotenkoordinaten. Die Nummerierung erfolgt automatisch nach Reihenfolge der Eingabe
- STATION 2 Stabermittlung → Bestimmung der Knoten und deren Verbindungen
- STATION 3 Übernahme der Eingabe der Kräfte
- STATION 4 Übernahme der Auflagerdaten und deren numerische Aufbereitung
- STATION 5 Aufbereitung des Systems
- STATION 6 Bildung der Einzelsteifigkeitsmatrizen und Addition zur Gesamtsteifigkeitsmatrix.
- STATION 7 Aufbau des Gesamtlastvektors
- STATION 8 Initialisierung der Festhaltungen und Reduzierung des Systems
- STATION 9 Lösung des Gleichungssystems und Bildung des Verschiebungsvektors
- STATION 10 Berechnung der Stabschnittgrößen und der Spannungen ($N / A$) aus dem Verschiebungsvektor

## 2.4 Ebenes Stabwerk

### 2.4.1 Theoretische Grundlagen

Die Berechnung des ebenen Stabwerks erfolgt wie bereits das Fachwerk ebenfalls auf der Grundlage der Methode der Finiten Elemente. Der Berechnungsablauf hierzu ist in [5], [8] beschrieben. Bezug ist der ebene, zweidimensionalen Raum. Die Achsen sind im Rechtssystem angeordnet, d.h. die Horizontalachse verläuft von links nach rechts positiv, die y-Achse verläuft vertikal von oben nach unten.

#### 2.4.1.1 Die Elementsteifigkeitsmatrix

Ebene Stabwerke besitzen an jedem Knotenpunkt drei Freiheitsgrade. Bei einem Stabwerk in der x-y-Ebene sind dies die Verschiebungen *u* und *v* sowie die Verdrehung φ.

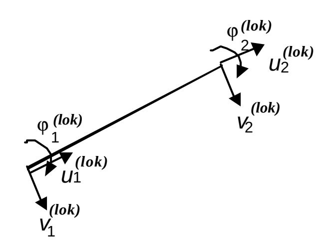
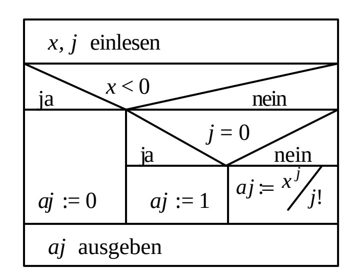



Bild 68. Knotenfreiheitsgrade des ebenen Stabwerks

Nach der Differentialgleichungsmethode ergibt sich für den geraden Stab mit konstantem Querschnitt und konstanter Biegesteifigkeit $E \cdot I$ die Koeffizientenmatrix nach [4] mit dem Reihen nach Rubin.

Dabei treten folgende Beiwerte auf:




$K, x, j$ einlesen

$bj := b := aj,\ s := j$

$s := s + 2$

$b := b * K * x^2 / s / (s-1)$

$bj := bj + b$

$|b| \le |bj| * e$

$bj$ ausgeben

Bild 69. Struktogramme für Beiwerte aj und bj

Die nichtlineare Elementsteifigkeitsmatrix $\underline{K}_{NL}$ ist damit folgendermaßen:

$$\underline{K}_{NL} = \begin{bmatrix} k_0 & 0 & 0 & -k_0 & 0 & 0 \\ 0 & k_3 & k_2 & 0 & -k_3 & k_2 \\ 0 & k_2 & k_1 & 0 & -k_2 & k_4 \\ -k_0 & 0 & 0 & k_0 & 0 & 0 \\ 0 & -k_3 & -k_2 & 0 & k_3 & -k_2 \\ 0 & k_2 & k_4 & 0 & -k_2 & k_1 \end{bmatrix}$$ Gl. (38)

mit den Parametern:
- $A$: Querschnittsfläche des Stabes [$m^2$]
- $A_Q$: Schubfläche [$m^2$]
- $E$: Elastizitätsmodul des Stabes [kN / $m^2$]
- $G$: Schubmodul [kN / $m^2$]
- $l$: Stablänge [m]
- $S_x$: Normalkraft im Stab [kN] (berechnet nach TH. I Ordnung bzw. Iterativ)

und den berechneten Werten für

$$g = \frac{1}{1 - \frac{S_x}{G \cdot A_Q}}$$ Gl. (39))

$$K = -g \cdot \frac{S_x}{E \cdot I}$$ Gl. (40))

und

$$C = g \cdot (b_2)^2 - b_1 \cdot \left( b_3 - \frac{E \cdot I}{G \cdot A_Q} \cdot b_1 \right)$$ Gl. (41)

ergeben sich die Koeffizienten der Matrix

$$k_0 = E \cdot A / l$$ Gl. (42)

$$k_1 = \frac{E \cdot I}{C} \cdot \left( l \cdot b_2 - b_3 + \frac{E \cdot I}{G \cdot A_Q} \cdot b_1 \right)$$

$$k_2 = \frac{E \cdot I}{C} b_2$$ Gl. (43)

$$k_3 = \frac{E \cdot I}{C} \cdot \frac{b_1}{g}$$ Gl. (44)

$$k_4 = \frac{E \cdot I}{C} \left( b_3 - \frac{E \cdot I}{G \cdot A_Q} b_1 \right)$$ Gl. (45)

### 2.4.1.2 Elementlastvektor

Als Lasten auf den Elementen wurden mehrere Lasttypen vorgesehen, wobei zunächst lediglich eine Trapezlast implementiert wurde. Diese setzt sich zusammen aus einer gleichförmigen Linienlast und einer Dreieckslast. Um die Theorie II. Ordnung mit in die Lasten einfließen zu lassen sind bei der Generierung der Elementlasten ebenfalls die Beiwerte nach [4] mit eingearbeitet. Damit lautet die Matrix für die Trapezlast:

$$C = g \cdot b_2^2 - b_1 \cdot bq_3 \qquad \text{Gl. (46)}$$

$$bq_j = b_j - \frac{E \cdot I}{G \cdot A_Q} b_j \qquad \text{Gl. (47)}$$

$$\Delta q = q_{Ende} - q_{Anfang} \qquad \text{Gl. (48)}$$

$$\underline{F}_{Trapez}^{Last}(E, G, A, A_Q, I, L, S_x, q_{Anfang}, q_{Ende}) = \begin{bmatrix} 0 \\ +Q_{Linienlast} + Q_{A-Dreieckslast} \\ -M_{Linienlast} - M_{A-Dreieckslast} \\ 0 \\ -Q_{Linienlast} - Q_{E-Dreieckslast} \\ +M_{Linienlast} + M_{E-Dreieckslast} \end{bmatrix} = \begin{bmatrix} N_A^0 \\ Q_A^0 \\ M_A^0 \\ N_E^0 \\ Q_E^0 \\ M_E^0 \end{bmatrix} \qquad \text{Gl. (49)}$$

$$Q_{Linienlast} = b_2 / b_1 \cdot g \cdot q_{Anfang} \qquad \text{Gl. (50)}$$

$$M_{Linienlast} = \frac{0.5 \cdot b_2 - b_3}{b_1} \cdot g \cdot q \qquad \text{Gl. (51)}$$

$$Q_{A-Dreieckslast} = \frac{b_3}{L \cdot b_1} \cdot g \cdot \Delta q \qquad \text{Gl. (52)}$$

Die Indizes $A$ und $E$ stehen dabei für die Belastung am Anfangsknoten und Endknoten des Elements, welche sich aus der allgemeinen Lösung der Differentiallösung ergeben.
Mit: $g$, $K$ wie vor,

$$Q_{E-Dreieckslast} = \frac{-b_2 - b_3 / L}{b_1} \cdot g \cdot \Delta q \qquad \text{Gl. (53)}$$

$$M_{E-Dreieckslast} = -\frac{b_3 - b_1 / b_2 \cdot b_4}{L} \cdot g \cdot \Delta q \qquad \text{Gl. (54)}$$

$$M_{A-Dreieckslast} = -\frac{bq_3 \cdot b_4 - bq_5 \cdot b_2}{L \cdot C} \cdot g \cdot \Delta q \qquad \text{Gl. (55)}$$

#### 2.4.1.3 Knotenlastvektor

Neben den Streckenlasten können in den Knotenpunkten auch einzelne Knotenlasten angesetzt werden. Dabei können Einzellasten in den globalen Richtungen vertikal und horizontal sowie Einzelmomente angreifen. Für den Fall, dass sich eine Einzellast in einem Feldabschnitt befindet ist das betreffende Feld zu Teilen, und die Last kann damit auf den eingefügten zusätzlichen Knoten aufgebracht werden.

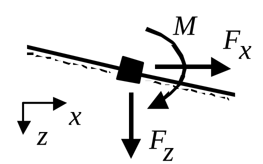



Bild 70. Knotenlasten

#### 2.4.1.4 Transformationsmatrix

Um die Elemente aus ihrer lokalen Richtung in die Globalkoordinaten umzuwandeln wird die folgende Transformationsmatrix verwendet:

$$\underline{T}(a) = \begin{bmatrix} \cos a & \sin a & 0 & 0 & 0 & 0 \\ -\sin a & \cos a & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos a & \sin a & 0 \\ 0 & 0 & 0 & -\sin a & \cos a & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Gl. (56)}$$

#### 2.4.1.5 Systemsteifigkeitsmatrix $\underline{K}_{ges}$

Aus den Steifigkeitsmatrizen wird die Systemsteifigkeitsmatrix, die auf alle Freiheitsgrade des statischen Systems bezogen ist, gebildet. Die Elementssteifigkeitsmatrizen werden hierzu auf die Systemsteifigkeitsmatrix systematisch aufaddiert. Die Verträglichkeitsbedingungen bleiben für die Einzelstäbe damit erhalten. Mit den drei Freiheitsgraden $u$, $v$ und $j$ an jedem Knoten bestimmt sich die Größe der Matrix zu:

$$n_{Matrix} = n_{Freiheitsgrade} \cdot n_{Knoten} \qquad \text{Gl. (57)}$$

mit $n_{Freiheitsgrade} = 3$ und $n_{Knoten} =$ Knotenanzahl des Gesamtsystems.

Die Transformation der Elemente auf die Globalkoordinaten geschieht mit der Gleichung:

$$\underline{K}_{el} = \underline{T}(a)^T \cdot K_{NL} \cdot \underline{T}(a) \qquad \text{Gl. (58)}$$

### 2.4.1.6 Lastvektor des Systems $\underline{F}_{ges}^{Last}$

Der Lastvektor des Gesamtsystems wird aus den an den Knotenpunkten angreifenden äußeren Lasten gebildet. Dazu wird zunächst ein Vektor erzeugt ,der die selbe Zeilenanzahl wie die Systemsteifigkeitsmatrix besitzt und anschließend die mit Kräften belegten Freiheitsgrade aufaddiert. Lasten aus schrägen Stäben müssen dabei über die Transformationsmatrix auf die Globalkoordinaten ausgerichtet werden.

$$\underline{F}_{el} = \underline{T}(a)^T \cdot \underline{F}_{Trapez}^{Last} \qquad \text{Gl. (59)}$$

### 2.4.1.7 Gleichungssystem

Mit den vorhanden Daten kann nun das Gleichungssystem für die gesamte Struktur gebildet werden. Es gilt:

$$\underline{K}_{ges} \cdot \underline{u}_{ges} = \underline{F}_{ges}^{Last} \qquad \text{Gl. (60)}$$

Dieses Gleichungssystem ist so jedoch nicht lösbar, da es zu viele Unbekannte besitzt. Mit den bekannten Verschiebungen an den Festhaltungen, die gleich Null sind, kann das Gleichungssystem auf ein lösbares System reduziert werden. Hierzu werden die Zeilen und Spalten die Festhaltungen besitzen eliminiert.
Es folgt

$$\underline{K}_{red} \cdot \underline{u}_{red} = \underline{F}_{red} \qquad \text{Gl. (61)}$$

umgestellt nach $\underline{u}_{red}$ lautet der Lösungsvektor:

$$\underline{u}_{red} = \underline{K}_{red}^{-1} \cdot \underline{F}_{red} \qquad \text{Gl. (62)}$$

hierzu wird die inverse Matrix von $\underline{K}_{red}$ gebildet.

### 2.4.1.8 Auflagerkräfte

Um die Auflagerkräfte zu erhalten wird wieder die Gesamtsteifigkeitsmatrix $\underline{K}_{ges}$ herangezogen. Diesmal wird diese dahingehend modifiziert, dass die Spalten der Koeffizienten an denen sich Festhaltungen befinden sowie die Zeilen die *keine* Festhaltungen besitzen eliminiert werden. Es entsteht eine Rechteckmatrix $\underline{K}_{red,Auflager}$ .Man erhält die Auflagerkräfte durch Multiplikation mit dem Verschiebungsvektor.

$$\underline{F}_{Auflager} = \underline{K}_{red,Auflager} \cdot \underline{u}_{red} \qquad \text{Gl. (63)}$$

### 2.4.1.9 Schnittgrößen am Einzelstab

Aus der bisherigen Berechnung sind die Verschiebungen an den Knotenpunkten vollständig bekannt, da an den Festhaltungen die Verschiebungen mit Null angesetzt werden können. Die Schnittgrößen ermitteln sich damit für den Einzelstab nach folgender Gleichung.

$$\begin{bmatrix} N_A \\ Q_A \\ M_A \\ N_E \\ Q_E \\ M_E \end{bmatrix} = \begin{bmatrix} k_0 & 0 & 0 & -k_0 & 0 & 0 \\ 0 & k_3 & k_2 & 0 & -k_3 & k_2 \\ 0 & k_2 & k_1 & 0 & -k_2 & k_4 \\ -k_0 & 0 & 0 & k_0 & 0 & 0 \\ 0 & -k_3 & -k_2 & 0 & k_3 & -k_2 \\ 0 & k_2 & k_4 & 0 & -k_2 & k_1 \end{bmatrix} \cdot \begin{bmatrix} u_A \\ v_A \\ j_A \\ u_E \\ v_E \\ j_E \end{bmatrix} - \begin{bmatrix} N_A^0 \\ Q_A^0 \\ M_A^0 \\ N_E^0 \\ Q_E^0 \\ M_E^0 \end{bmatrix} \qquad \text{Gl. (64)}$$

bzw.:

$$\underline{F}_{Element}^{global} = \underline{K}_{NL} \cdot \underline{u}_{Element} - \underline{F}_{Element}^{Last} \qquad \text{Gl. (65)}$$

Um die Schnittgrößen rechtwinklig zu den Elementen des unverformten Systems zu erhalten werden die globalen Größen wiederum transformiert.

$$\underline{F}_{Element}^{lok} = \underline{T}(a)^T \cdot \underline{F}_{Element}^{global} \qquad \text{Gl. (66)}$$

### 2.4.2 Umsetzung in JAVA

#### 2.4.2.1 Klassenstruktur

Verwendete Klassen und deren interner Zusammenhang:

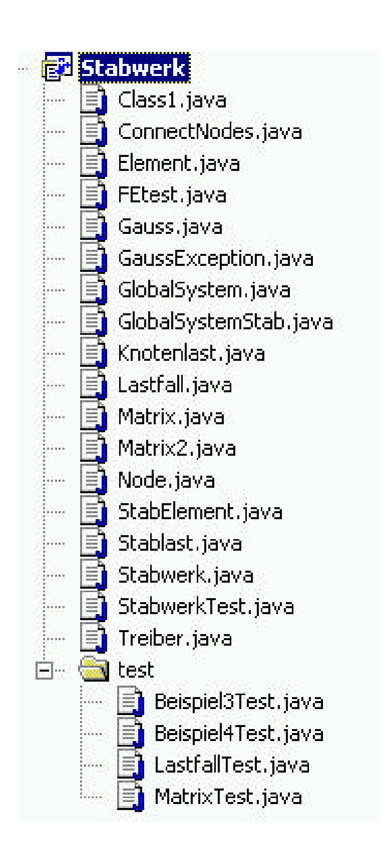



Bild 71. Implementierte Klassen in Stabwerk.dll

#### 2.4.2.2 Objekt-Schnittstellen

Die Klasse *Stabwerk.java* definiert die Schnittstelle zu den Anwenderprogrammen, welche die COM DLL verwenden (zum Beispiel Mathcad oder Excel). Die Ein- und Ausgabeoperationen orientieren sich im Stil der Namen und Parameter an der Schnittstellenklasse zur Fachwerksberechnung um eine einheitliche Schnittstelle aller Berechnungsmodule zu erreichen. Die Klasse Stabwerk speichert übergebene Daten intern ab und ermöglicht Zugriff auf bereits eingegebene Daten, wie auch auf die Ergebnisse der Berechnung.

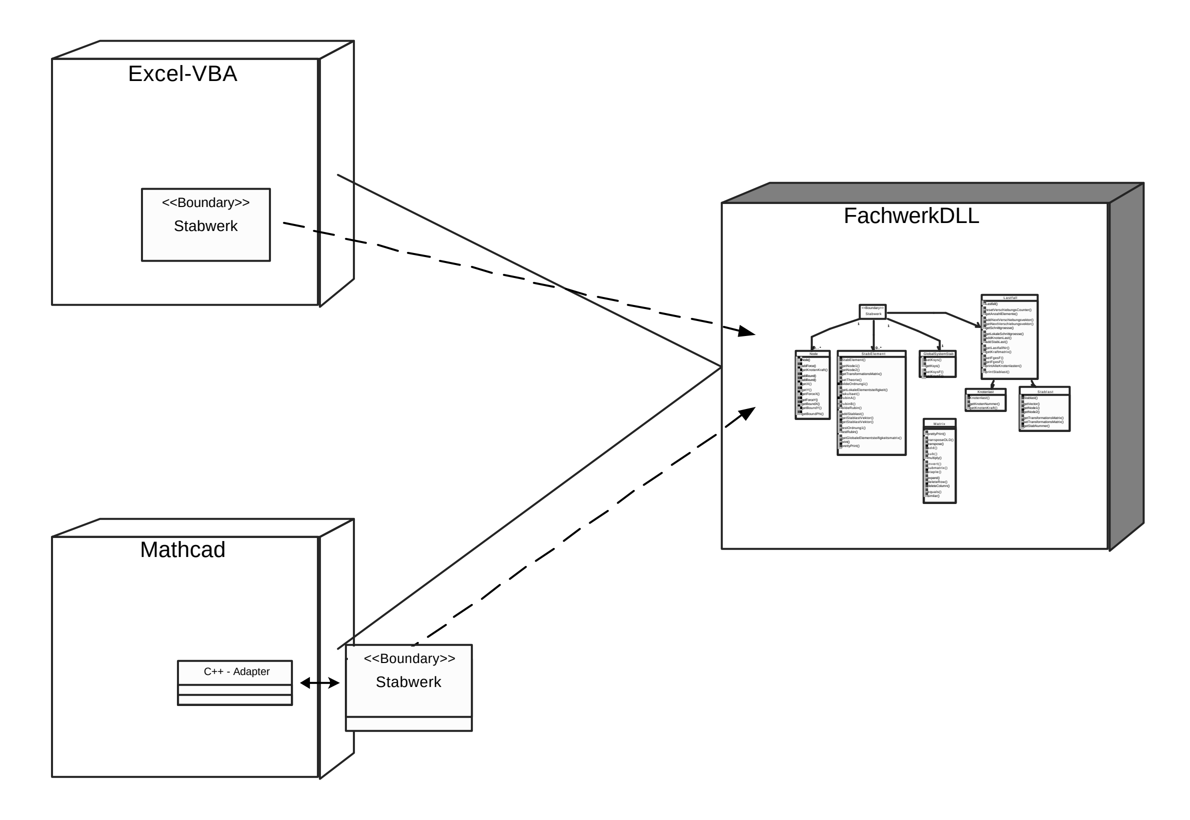



Bild 72. Zugriffsstruktur auf die einzelnen Komponenten

Daten zu Knoten, Stäben, sowie Kräften werden mit Hilfe folgender Operationen eingegeben:

- public Stabwerk( )
  Default Konstruktor – legt das Objekt an. Keine weiteren Daten nötig

- public int addNode(double x,double y)
  Knotenkoordinaten in [m]. Nummerierung erfolgt
  automatisch nach Reihenfolge der Eingabe
  x, y: Koordinatenwerte in Meter
  Return: Nummer des Knotens

- public int addElem(int node1, int node2, double A, double E,
  double I, double AQ, double G,
  int unterteilungen)
  Eingabe der Stäbe. Welcher Knoten ist mit welchem verbunden? Die Nummerierung erfolgt wieder intern nach der Reihenfolge der Eingabe, beginnen mit 1.
  node1, node2: Knotennummer

| | |
|---|---|
| A: | Stab Fläche |
| E: | Stab E-Modul |
| I: | Trägheitsmoment |
| AQ: | Schubfläche |
| G: | Schubmodul |
| Unterteilungen: | Anzahl der Unterteilungen aus denen das Element bestehen soll ( 1 für keine, 2 für 2 Teile, ...) |
| Return: | Nummer des Elements(1,2,…n) |

- `public int addKnotenLast(int knotenNr,int lastfallNr, double x, double y)`
  Eingabe der Knotenkräfte in kN
  knotenNr: Nummer des Knotens beginnend mit 1
  lastfallNr: Lastfall für den die Kraft gelten soll (1..n)
  x, y: Kräfte in x- und y-Richtung [kN]

- `public int addStabLast (int stabNr,int lastfallNr, double qAnfang, double qEnd)`
  Eingabe der Stablasten in kN
  stabNr: Nummer des Stabs, beginnend mit 1
  lastfallNr: Lastfall für den die Kraft gelten soll (1..n)
  qAnfang, qEnd: Anfangs- und Endkraft

- `public int addTemperaturLast(int stabNr,int lastfallNr, double deltaT, double alphaT, double h)`
  Eingabe der Temparaturlasten ( nicht implementiert)

- `public int addBound(int num, int x, int y)`
  Eingabe der Auflagenfixierung. Welcher Knoten ist in x und/oder y und/oder phi gelagert?
  Initialisiert wird jeder Knoten mit 1 (→ nicht gelagert).
  num: Knotennummer (1..n)
  x, y, phi Auflage in x/y/phi 1 → nicht gelagert,
  -1 → gelagert

Die Berechnung wird anschließend mit folgender Operation angestoßen:

- `public int solve(int theorie )`
  Berechnet das Stabwerk
  theorie: 1 → Berechnung nach Theorie I. Ordnung
  2 → Berechnung nach Theorie II. Ordnung

Nach der Berechnung stehen die Daten bereit und können mit Hilfe der Ausgabe Schnittstelle der Reihe nach ausgelesen werden.

Dabei stehen pro Lastfall folgende Werte zur Verfügung:

- `public double getN(int stabNr,int lastfallNr,int teilung)`
  Normalkraft an der entsprechenden Stabteilung
  stabNr: Nummer des Stabelements [1..n]
  lastfallNr: Nummer des Lastfalls [1..n]
  teilung: Nummer der Teilung [0..Anzahl Teilungen]

- `public double getQ(int stabNr,int lastfallNr,int teilung)`

  Querkraft an der entsprechenden Stabteilung

- `public double getM(int stabNr,int lastfallNr,int teilung)`
  Moment an der entsprechenden Stabteilung

- public double knotenVerschiebungX(int knotenNr,int lastfallNr)
  X-Verschiebung des Elements
- public double knotenVerschiebungY(int knotenNr,int lastfallNr)
  Y-Verschiebung des Elements

Zusätzlich können noch die Minima und Maxima mit folgenden Methoden ausgelesen werden:

- public double getN_MIN(int stabNr,int teilung)
  Minimale Summe der auftretenden Normalkräfte an der Stabteilung
- public double getQ_MIN(int stabNr,int teilung)
  Minimale Summe der auftretenden Querkräfte an der Stabteilung
- public double getM_MIN(int knotenNr,int teilung)
  Minimale Summe der auftretenden Momente an der Stabteilung
- public double knotenVerschiebungX_MIN(int knotenNr)
  Minimale Summe der auftretenden X-Verschiebungen am Knoten
- public double knotenVerschiebungY_MIN(int knotenNr)
  Minimale Summe der auftretenden Y-Verschiebungen am Knoten
- public double getN_MAX(int stabNr,int teilung)
  Maximale Summe der auftretenden Normalkräfte an der Stabteilung
- public double getQ_MAX(int stabNr,int teilung)
  Maximale Summe der auftretenden Querkräfte an der Stabteilung
- public double getM_MAX(int stabNr,int teilung)
  Maximale Summe der auftretenden Momente an der Stabteilung
- public double knotenVerschiebungX_MAX(int knotenNr)
  Maximale Summe der auftretenden X-Verschiebungen am Knoten
- public double knotenVerschiebungY_MAX(int knotenNr)
  Maximale Summe der auftretenden Y-Verschiebungen am Knoten

Des weiteren stehen folgende Methoden zur Verfügung:

- public int getNumberNodes()
  Liefert die Anzahl der eingegebenen Knoten zurück
- public double getNodeX(int pos)
  Liefert die x-Koordinate des Knoten pos zurück
- public double getNodeY(int pos)
  Liefert die y-Koordinate des Knotens pos zurück

- public int getNumElem()
  Liefert die Anzahl an Stäben im System zurück

```
<<Boundary>>
Stabwerk

Stabwerk()
addNode(x : double, y : double) : int
addElem(node1 : int, node2 : int, myA : double, myE : double, myI : double, myAQ : double, myG : double, unterteilungen : int) : int
addKnotenLast(knotenNr : int, lastfallNr : int, x : double, y : double) : int
addStabLast(stabNr : int, lastfallNr : int, qAnfang : double, qEnd : double) : int
addTemperaturLast(stabNr : int, lastfallNr : int, deltaT : double, alphaT : double, h : double) : int
addBound(num : int, x : int, y : int) : int
getNumberNodes() : int
getNodeX(pos : int) : double
getNodeY(pos : int) : double
getNumElem() : int
solve(theorie : int) : int
getN(stabNr : int, lastfallNr : int, teilung : int) : double
getQ(stabNr : int, lastfallNr : int, teilung : int) : double
getM(stabNr : int, lastfallNr : int, teilung : int) : double
knotenVerschiebungX(knotenNr : int, lastfallNr : int) : double
knotenVerschiebungY(knotenNr : int, lastfallNr : int) : double
getN_MIN(knotenNr : int, teilung : int) : double
getQ_MIN(knotenNr : int, teilung : int) : double
getM_MIN(knotenNr : int, teilung : int) : double
knotenVerschiebungX_MIN(knotenNr : int) : double
knotenVerschiebungY_MIN(knotenNr : int) : double
getN_MAX(knotenNr : int, teilung : int) : double
getQ_MAX(knotenNr : int, teilung : int) : double
getM_MAX(knotenNr : int, teilung : int) : double
knotenVerschiebungX_MAX(knotenNr : int) : double
knotenVerschiebungY_MAX(knotenNr : int) : double
```

Bild 73. Methoden der Klasse *Stabwerk*

### 2.4.2.3 Interne Schnittstellen

#### 2.4.2.3.1 Aufbau und Design

Die Berechnung der einzelnen Zwischenschritte geschieht verteilt auf die zuständigen Klassen. So werden zum Beispiel die Elementsteifigkeitsmatrizen innerhalb der Klasse Stabelement berechnet und gespeichert. Dies geschieht allerdings noch nicht beim Anlegen der Elemente, sondern erst beim Start der Berechnung mit *solve()*, da erst dann feststeht, welche Theorie der Berechnung zu Grunde liegt. Durch die Verteilung der Berechnungsschritte auf die unterschiedlichen Klassen, erreicht man eine übersichtlichere Schnittstellenklasse (*Stabwerk.java*). Zusätzlich können bei Änderungen einzelne Klassen einfacher ausgetauscht werden ohne das Zusammenspiel der Klassen zu stören. Allerdings liegt der Hauptalgorithmus, der regelt wie die einzelnen Schritte aufgerufen werden müssen, innerhalb der solve-Methode. Berechnungsfunktionen innerhalb der COM Klasse wurden als private Operationen definiert. Dies erhöht die Übersichtlichkeit ohne die Kapselung der Daten zu durchbrechen. Aus dem vormals erwähnten Programm "Fachwerke" konnten innerhalb des Stabwerks für Biegebalken keine Klassen übernommen werden, da die Berechnungsschritte, wie auch die interne Struktur der Daten, zu unterschiedlich sind. Die einzige Klasse, die von einem anderen System übernommen wurde, ist die Klasse Matrix, die Berechnungen für Matrizen zur Verfügung stellt. Allerdings musste diese Klasse auch deutlich erweitert werden. Später wurde noch eine weitere Matrix Klasse (*Matrix2.java*) eingeführt, die aus einer anderen Quelle stammt. Leider sind beide Klassen nicht vollkommen fehlerfrei. Fehlerhafte Methoden, wie zum Beispiel *Matrix.invert()*, sind im Quelltext als solche gekennzeichnet.

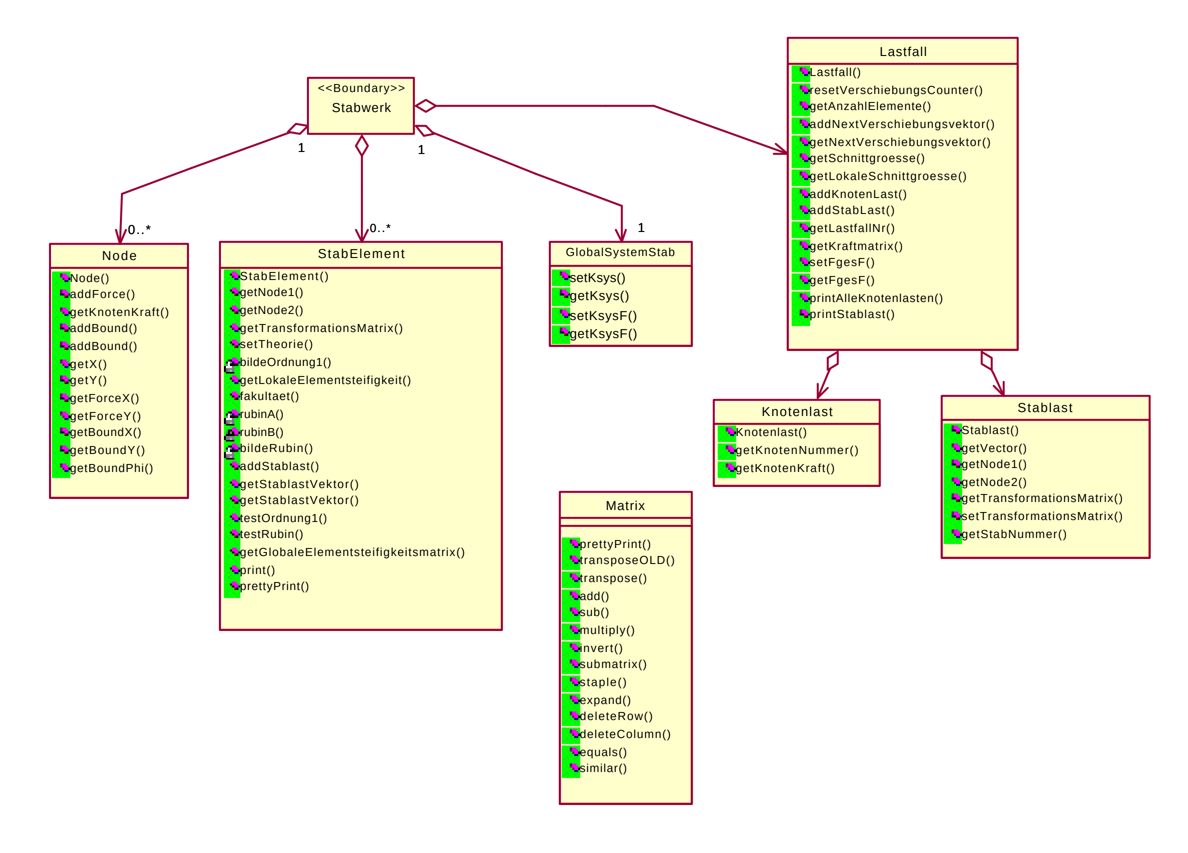



Bild 74. UML Klassenstruktur des Fachwerk Programms

## 2.4.2.3.2 Test

Um die einzelnen Berechnungsschritte und Module zu testen, wurde das jUnit Test-Framework von Kent Beck und Erich Gamma eingesetzt.

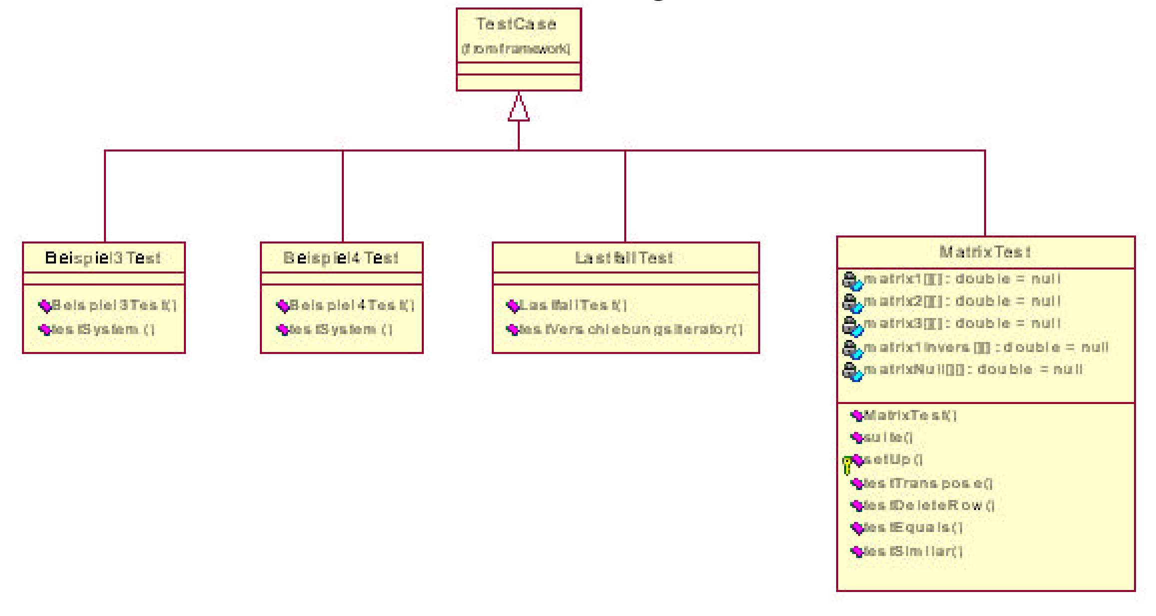



Bild 75. jUnit Test-Framework für das Stabwerkprogramm

Dabei handelt es sich um ein Framework für Komponententests, die während der gesamten Entwicklungszeit ausgeführt werden können. Somit wurden die wichtigsten Methoden und Klassen ständig getestet. Bei weiteren Änderungen am System lassen sich mit den

vorhandenen Testfällen eventuelle Auswirkungen auf die bestehenden Funktionen überprüfen. Die Komponententests enthalten nicht nur Überprüfungsmethoden für die Klassen und Methoden, sondern auch zwei Testfälle, die ein komplettes System berechnen und die Ergebnisse überprüfen. Somit stehen also auch fachliche Funktionstests zur Verfügung.

Die Klassenbibliothek des jUnit Frameworks liegt dem Quellcode bei. Die ausführbare Testsuite *static suite( )* befindet sich in der Klasse MatrixTest. Eine Beschreibung zur wie jUnit anzuwenden ist gibt es auf der Internetseite mit der Adresse www.jUnit.org.

#### 2.4.2.4 Funktionsumfang

Die DLL berechnet zweidimensionale Stabwerke auf Basis von Biegebalken mit drei Freiheitsgraden. Zur Berechnung der Stabwerke werden die Knoten, die Elemente, die Lasten, sowie alle zugehörigen Daten eingegeben. Lasten können verschiedenen Lastfällen zugeordnet werden. Die Berechnung des gesamten Systems erfolgt nach Theorie I. Ordnung. Eine Berechnung des Systems mittels Theorie II Ordnung ist vorgesehen und für Teile des Systems implementiert. Dabei werden die Gesamtsteifigkeitsmatrix, die Elementsteifigkeiten sowie die Kräfte nach Theorie II Ordnung berechnet. Eine Iteration der Normalkraft erfolgt aber noch nicht.

Berechnet werden die Schnittgrößen an den Stabteilungen. Dabei werden die Normalkraft, die Querkraft, sowie das Moment an verschiedenen Stabteilungen ausgegeben. Die Anzahl der Stabteilungen pro Stab kann frei gewählt werden. Dabei werden die Schnittgrößen jedoch nur bezogen auf die Stablängsrichtung gerechnet, nicht aber auf das verformte System umgerechnet.

Zusätzlich werden von der DLL die minimalen und maximalen Schnittgrößen für jede Teilung in allen Elementen des Systems berechnet.

## 2.5 Stahlbetonbau

### 2.5.1 Theoretische Grundlagen

Für die Bemessung von Stahlbetonbauteilen stehen zur Zeit mehrere Normen und Vorschriften zur Verfügung. Im Moment gelten neben der alten DIN 1045 von 11.88 die Nachweise nach Eurocode 2 (Stahl- und Spannbetonbauteile) sowie die Nachweise nach der neuen DIN 1045-1 von 02.2001. Es wurden daher Funktionen entwickelt, die auf dem Eurocode als auch auf der neuen DIN Norm basieren, wobei zum Teil unterschiedliche Konzepte Anwendung fanden.

#### 2.5.1.1 Biegebemessung

Die Biegebemessung von Stahlbetonbauteilen wurde für den EC 2 und die DIN 1045-1 getrennt geführt. In beiden Fällen wurde ein Rechteckquerschnitt betrachtet.

***Bemessung nach DIN 1045-1:***
Die Bemessung erfolgt mit einem dimensionsechten[4] Verfahren. Es wird das Ebenbleiben des Betonquerschnitts vorausgesetzt, d.h. die Dehnungen im Querschnitt verlaufen linear. Als zulässige Dehnungen liegen die Werte der Tabelle des Abschnitts 9.1.7 der DIN 1045-1 zugrunde. Dabei sind die Stahldehnungen auf 25 %o beschränkt. Für die Ermittlung der Bemessungsbeiwerte ω, ξ, und ζ wird nach dem Momentengleichgewicht der inneren und äußeren Kräfte gesucht.

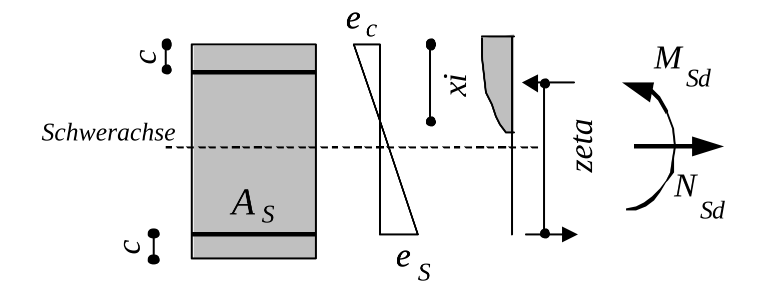



Bild 76. Bezeichnungen für den Betonquerschnitt nach DIN 1045-1

Dabei wird die Druckzonenhöhe so lange vergrößert bis sich Gleichgewicht einstellt. Die Bemessung der Biegezugbewehrung erfolgt mit diesen Werten. Die aufnehmbare Druckspannungen des Stahlbetons wird dabei nach den Formeln in DIN 1045-1 ermittelt. Für Betone bis C50/60 gilt:

$$f_{cd} = a \cdot f_{ck} / g_c \qquad \text{Gl. (67)}$$

mit $a$ als Faktor zur Berücksichtigung der Langzeitwirkung nach [9] Seite 5.31

$$s_c = -1000 \cdot \left(e_c + 250 \cdot e_c^2\right) \cdot f_{cd} \qquad \text{Gl. (68)}$$

[4] Die Eingabeparameter können in beliebigen Einheiten (m, cm, mm) gemacht werden

ab C55/67 gilt:

$$s_c = -1000 \cdot \left[ 1 - \left( 1 - \left( \frac{e_c}{e_{c2}} \right) \right)^n \right] \cdot f_{cd} \qquad \text{Gl. (69)}$$

Um die aufnehmbare Betondruckkraft zu errechnen werden die Betondruckspannungen über die Höhe ξ integriert:

$$F_1 = \int_0^{x_1} s_c \, dx \qquad \text{Gl. (70)}$$

Da die Betondruckkraft mit der Stahlzugkraft im Gleichgewicht steht, kann hieraus der innere Hebelarm ermittelt werden. Den Abstand der Betondruckkraft von der Spannungsnullline errechnet sich durch

$$e_c = \frac{1}{F_1} \cdot \int_0^{x_1} (s_c \cdot x) dx \qquad \text{Gl. (71)}$$

Damit ergibt sich der Hebelarm der inneren Kräfte zu:

$$V = h - c - x_1 + e_c \qquad \text{Gl. (72)}$$

Falls die Druckzonenhöhen die nach der Norm vorgeschrieben Werte überschreitet, ist Druckbewehrung erforderlich. In diesem Falle werden die in der DIN 1045-1 zulässigen Werte für die Druckzonenhöhe eingefroren und die aufnehmbare Betondruckkraft ermittelt. Daraus ermittelt man nun das aufnehmbare Moment mit dem zugehörigen Hebelarm, welches vom vorhanden Bemessungsmoment abgezogen wird. Das Differenzmoment muss nun durch ein Kräftepaar mit dem Hebelarm des Bewehrungsabstandes von Druck- und Zugbewehrung aufgenommen werden. Steht dieses Kräftepaar fest, kann daraus unter Berücksichtigung der spezifischen Streckgrenze des Bewehrungsstahls und des erforderlichen Sicherheitsbeiwertes die Druckbewehrung berechnet werden. Die Zugbewehrung ergibt sich als Summe der Bewehrung aus erforderlicher Bewehrung ohne Druckbewehrung und der Druckbewehrung.

***Bemessung nach EC 2:***

Die Bemessung nach EC 2 erfolgt nach [10] mit der Bemessungstafel 3a. Diese beinhaltet dimensionsgebundene Beiwerte für den Rechteckquerschnitt ohne Druckbewehrung für Biegung mit Längskraft, basierend auf der Betonstahlsorte BSt 500 und dem Sicherheitsbeiwert $g_S = 1.15$ nach [11].

Für die Anwendung der Bemessungstafeln wird das Bemessungsmoment

$$M_{Sds} = M_{Sd} - N_{Sd} \cdot z_{s1} \qquad \text{Gl. (73)}$$

mit

$M_{Sd}$ : maßgebende Moment auf Desing-Ebene [kNm],
$N_{Sd}$ : maßgebende Normalkraft [kN]
$z_{s1}$ : Abstand der Zugbewehrung von der Schwerachse [m]

berechnet, um den Eingangswert $k_d$ in die Tafel zu erhalten.

$$k_d = \frac{d}{\sqrt{M_{Sds}/b}}$$ Gl. (74)

mit

$d$: statische Höhe [cm]
$b$: Breite des Betonquerschnitts [m]

Die Tafel liefert den Wert $k_s$ mit dem die erforderliche Bewehrung berechnet wird.

$$A_{S,erf} = k_s \cdot \frac{M_{Sds}}{d} + \frac{N_{Sd}}{43.5}$$ Gl. (75)

In die Tafel sind die Werte für die Betonfestigkeitsklassen C12/15 bis C50/60 eingearbeitet.

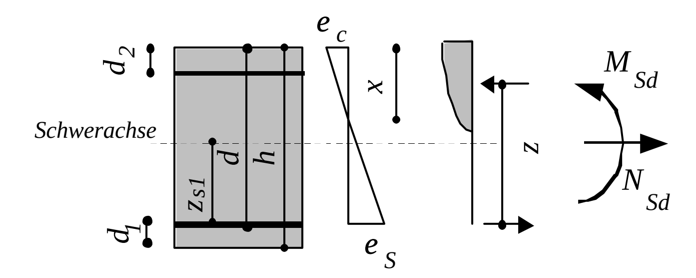



Bild 77. Bezeichnungen für den Betonquerschnitt nach EC2

### 2.5.1.2 Schubbemessung

Die Ermittlung der Schubbewehrung erfolgt auf der Grundlage der DIN 1045-1 nach den Formeln der [9] für den nicht gevouteten Querschnitt mit parallelen Rändern. Eine Bemessung ist für Normal- und Leichtbetone vorgesehen. Für die Bemessung stehen dabei aufgrund der Softwareeinschränkungen mehrere Funktionen zur Verfügung. Die Parameter, welche dabei immer eingegeben werden müssen, sind: Bemessungsquerkaft, Querschnittsbreite und statische Nutzhöhe sowie die Betondruckfestigkeit. In den weiteren Funktionen kommen Parameter für Leichtbetone, Neigung der Schubbewehrung, weitere Nennfestigkeiten für Betonstähle, Normalspannungen aus Vorspannung oder Zwang, veränderte Sicherheitsbeiwerte, Querschnitte der Biegezug- und –druckbewehrung sowie Abstand einer auflagernahen Einzellast und Variationen aus diesen Parametern vor.

Zunächst wird im Programm überprüft, ob der Schubwiderstand des Bauteils ohne Schubbewehrung überschritten wird.

$$V_{Rd,ct}(h_1, A_{Sl}, f_{ck}, s_{cd}, b_w, d)$$

$$V_{Rd,ct} = \left[0.10 \cdot h_1 \cdot k \cdot (100 \cdot r_l \cdot f_{ck})^{1/3} - 0.12 \cdot s_{cd}\right] \cdot b_w \cdot d$$ Gl. (76)

Hierin sind:

$h_1$ : Tragfähigkeitsbeiwert;
$h_1 = 1.0$ für Normalbeton
$h_1 = 0.40 + 0.60 \cdot (r/2200)$ für Leichtbeton (ρ: Trockendichte [kg/m³]

κ: Beiwert für den Einfluss der Bauteilhöhe d (mit d [mm])

$$k = 1 + \sqrt{200/d} \le 2$$ Gl. (77)

$b_w$: Querschnittsbreite [m]
$s_{cd}$: Längsspannungen infolge Last oder Vorspannung [MN / m²]
$f_{ck}$: charakteristische Betondruckspannung [MN / m²]
$r_1$: Längsbewehrungsgrad

$$r_1 = A_{sl} / b_w \cdot d \le 0.02$$ Gl. (78)

$A_{sl}$: Fläche der Längsbewehrung, die mindestens mit $(d + l_{b,net})$ über den betrachteten Schnitt hinausgeführt wird.

Ist dies der Fall, wird die Betondruckstrebenneigung ermittelt und die maximal aufnehmbare Querkraft daraus ermittelt. Ist dies nicht der Fall, wird der erforderliche Bewehrungsquerschnitt zu Null gesetzt und der weitere Berechnungsablauf abgebrochen.

Bestimmung der Druckstrebenneigung $J$ :

$$\cot J \le \frac{(1.2 - 1.4 \cdot s_{cd} / f_{cd})}{(1 - V_{Rd,c} / V_{Sd})} \begin{cases} \ge 1.0 \\ \le 3.0 \end{cases}$$ Gl. (79)

(für Normalbeton, bei Leichtbeton gilt $\cot J \le 2.0$)

mit $V_{Rd,c}(f_{ck}, f_{cd}, s_{cd}, h_1, b_{ct})$

$$V_{Rd,c} = \left[ h_1 \cdot b_{ct} \cdot 0.10 \cdot f_{ck}^{1/3} \cdot \left( 1 + 1.2 \cdot \left( s_{cd} / f_{cd} \right) \right) \right] \cdot b_w \cdot z$$ Gl. (80)

wobei $b_{ct}$ = 2.4 ist.

Ermittlung des Bemessungswiderstandes $V_{Rd,\max}(f_{cd}, b_w, z, a_c, a, J)$:

$$V_{Rd,\max} = a_c \cdot f_{cd} \cdot b_w \cdot z \cdot \frac{\cot J + \cot a}{1 + \cot^2 J}$$ Gl. (81)

mit

$$f_{cd} = a \cdot f_{ck} / g_c$$ Gl. (82)

(Bemessungswert der Betondruckfestigkeit)

$$a_c = 0.75 \cdot h_1 \quad mit \quad \begin{cases} h_1 = 1.0 & \text{für Normalbeton} \\ h_1 = 0.40 + 0.60 \cdot r / 2200 & \text{für Leichtbeton} \end{cases}$$ Gl. (83)

$z \approx 0.90 \cdot d$ Hebelarm der inneren Kräfte
$a$: Winkel zwischen Schubbewehrung und Bauteilachse

Überschreitet die Bemessungsquerkraft die aufnehmbare Querkraft, wird eine Fehlermeldung ausgegeben. Wenn Schubbewehrung erforderlich wird und die aufnehmbare Querkraft nicht überschritten wird, erfolgt die Bemessung.

Grundlagen der Schubbemessung sind folgende Formeln und Beziehungen:

$a_{sw,erf}\,(V_{Sd}\,,\,f_{yd}\,,z,a\,,J\,)$

$$erf_a_{sw} = \frac{V_{Sd}\cdot 10^{-3}\cdot 100^{2}}{f_{yd}\cdot z\cdot\left(\cot\left(\Theta_{Grad}\cdot\frac{p}{180}\right)+\cot\left(a\cdot\frac{p}{180}\right)\right)\cdot\sin\left(a\cdot\frac{p}{180}\right)} \qquad \text{Gl. (84)}$$

mit

$$f_{yd} = f_{yk} \,/\, g_S \qquad \text{Gl. (85)}$$

## 2.5.2 Programmierung in JAVA

### 2.5.2.1 Biegebemessung nach DIN 1045-1

Die Objektbibliothek *concretR.DLL* für die Biegebemessung nach DIN 1045-1 enthält die beiden Klassen *hilfsfunktion* und *Bemessung1*, Bild 3.

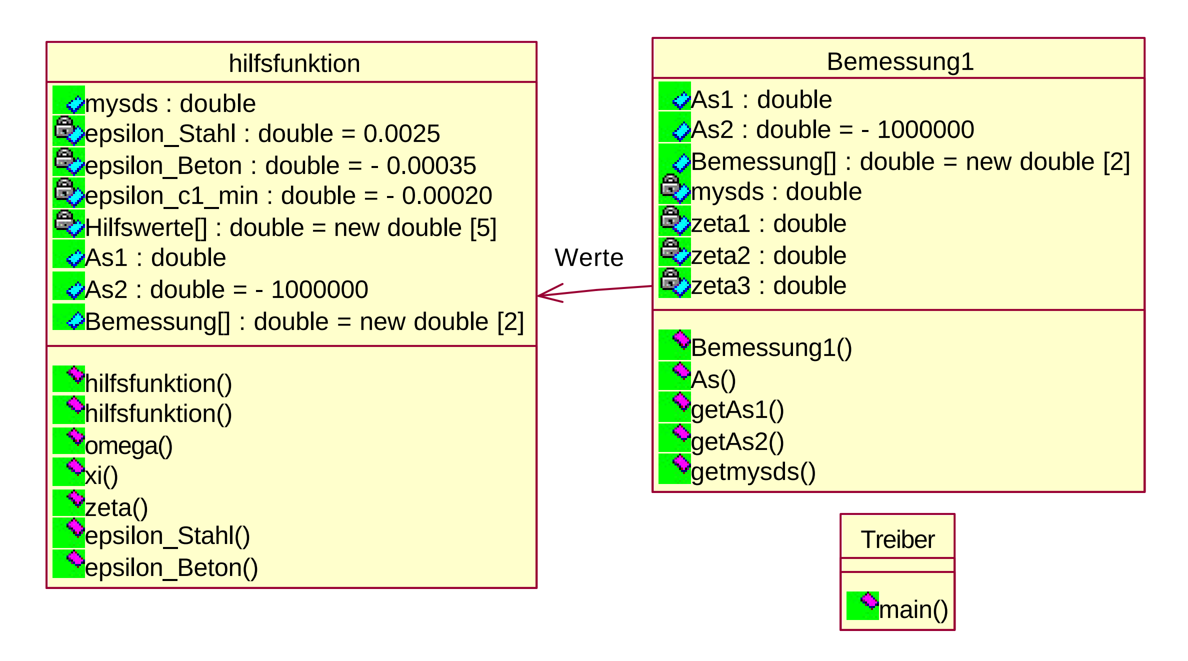



Bild 78. UML-Klassendiagramm von *concretR.DLL*

### Klasse *hilfsfunktion*

In der Klasse *hilfsfunktion* sind die Iteration für die Bemessungswerte nach dem dimensionslosen Verfahren enthalten. Es werden darin die Werte $w$, $x$, $z$, $e_{c2}$, und $e_{S1}$ berechnet. Die Iteration wird in Erhöhungsschritten der Druckzonenhöhe von 0.001 durchgeführt.

**Attribute:**

```
double mysds                          // Bemessungswert
double epsilon_Stahl = 0.0025         // Ergebniswert Stahldehnung
double epsilon_Beton = -0.00035       // Ergebniswert Betonstauchung
double epsilon_c1_min = -0.00020      // minimale Betonstauchung bei der fc erreicht wird
double[ ] Hilfswerte = new double[5]  // Ergebnisfeld für die Beiwerte
double[ ] Bemessung = new double[2]   // Ergebnisfeld Zug- u. Druckbewehrung
public double As1                     // Ergebniswert Zugbewehrung
public double As2 = -1000000          // Ergebniswert Druckbewehrung
```

**Methoden**

Konstruktor:

```
public hilfsfunktion()

public double[ ] hilfsfunktion (double mysds)
```

Der Bemessungswert *mysds* der Biegeschnittgröße wird übergeben und daraus die Bemessungsbeiwerte errechnet. Diese sind im Ergebnisfeld Hilfswerte enthalten.

```
public double omega(double mysds)

public double xi(double mysds)               // Übergibt die Druckzonenhöhe

public double zeta(double mysds)             // Übergibt die Höhe des inneren Hebelarmes

public double epsilon_Stahl(double mysds)    // Übergibt die zugehörige Stahldehnung

public double epsilon_Beton(double mysds)    // Übergibt die zugehörige Betonstauchung
```

**COM-Klasse *Bemessung1***

Die Klasse *Bemessung1* berechnet die erforderliche Bewehrung. Werden die Werte für

mysds_lim1 = 0.296 // bei Betonfestigkeiten bis 50 MN/m$^2$
bzw.
mysds_lim2 = 0.372 // für höhere Festigkeiten

überschritten, so werden diese Werte eingefroren und die erforderliche Druckbbewehrung ermittelt.

**Attribute:**

```
double As1                            // Ergebniswert Zugbewehrung
double As2 = -1000000                 // Ergebniswert Druckbewehrung
```

```
double[] Bemessung = new double[2]          // Ergebnisfeld Zug- u. Druckbewehrung
double mysds, zeta1, zeta2, zeta3;
```

**Methoden**

Konstruktor:

```
public Bemessung1()
public int As(double b, double h, double c, double Msd, double Nsd, double fcd)
```

Parameter:

```
    b              // Bauteilbreite
    h              // Bauteilhöhe
    c              // Bewehrungsabstand vom Rand
    Msd            // Bemessungsmoment
    Nsd            // Bemessungsnormalkraft
    fcd            // Betondruckfestigkeit
```

Zunächst wird *mysds* berechnet, mit dem die Hilfswerte berechnet werden. Es werden dann mit Hilfe von *hilfswerte.omega(mysds)* die Bewehrung für As1 und As2 bestimmt. Falls keine Druckbewehrung erforderlich ist bleibt As2 = 10000000.

```
public double getAs1()      // Methode dient zum Auslesen der berechneten Werte

public double getAs2()      // Methode dient zum Auslesen der berechneten Werte

public double getmysds()    // Methode dient zum Auslesen von mysds
```

### 2.5.2.2 Biegebemessung nach EC 2

Die Objektbibliothek *As.DLL* besteht aus lediglich einer Klasse.



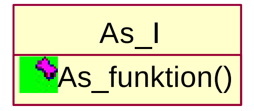




Bild 79. UML Diagramm der Klasse AS_I

**Die COM-Klasse *As_I***

Die Klasse *As_I* beinhaltet sämtliche Daten der Bemessung wie z.B. die kd-Werte mit den zugehörigen ks-Werten. Diese sind in der Klasse selbst in zwei- bzw. eindimensionalen Arrays abgespeichert.

k_d [Betongüte][kd-Wert]

kss [zugehörige ks-Werte]

```
public String As_funktion(hr, d, b, N, M, c_fest)
```

**Attribute:**

```
hr        // Höhe des Rechtechquerschnitts
d         // statische höhe
b         // Querschnittsbreite
N         // Normalkraft
M         // Moment
c_fest    // Betonbenennung
```

Zunächst werden die benötigten Bemessungsparameter

```
zs= hr/2-(hr-d);
ms=M-N*zs/100;
b=b/100;
kd=d/Math.sqrt(ms/b);
```

ermittelt.

Für die Betonbenennungen c_fest sind die Werte

"12/15", "16/20", "20/25", "25/30", "30/37", "35/45", "40/50", "45/55", "50/60"

vorgesehen.

Diese Strings werden verglichen und einem Integerwert zugewiesen, der als Eingangswert für das *kd* -Array gilt.

Für den Fall, dass die Betonfestigkeiten nicht definiert ist , ist die Fehlermeldung

"Betonfestigkeitsklasse nicht definiert!"

vorgesehen. Ist kein passender *kd* -Wert definiert, wird

"Druckbewehrung erforderlich!"

ausgegeben.

Ansonsten ergibt sich durch Interpolation der benachbarten *kd* -Werte der gesuchte *ks* -Wert. Hiermit ermittelt sich die erforderliche Biegezugbewehrung nach der Gleichung:

```
AS=ks*(ms/d)+N/43.5
```

Sie wird vor einer Ausgabe in einen String umgewandelt um allen Ergebnisarten im selben Format zu haben.

### 2.5.2.3 Schubbemessung nach DIN 1045-1

Die in Java programmierte Bibliothek *StahlBeton.dll* besteht aus zwei Klassen:

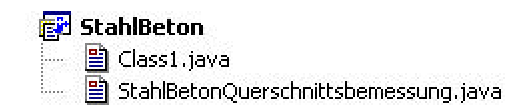



Bild 80. Projektdateien der Bibliothek *StahlBeton.dll*

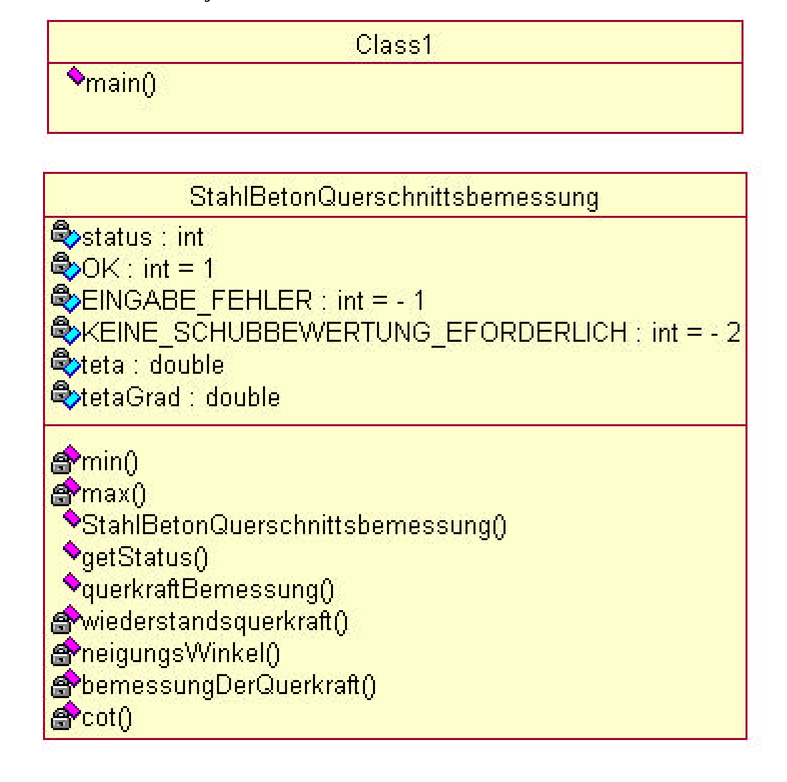



*Bild 81.* UML Diagramm der Klassen *Class1* und *StahlbetonQuerschnittsbemessung*

#### Die Klasse *Class1*

Bei der Klasse *Class1* handelt es sich lediglich um eine Hilfsklasse zu Debug- und Testzwecken. Dazu dient die *main()* – Methode.

#### Die COM-Klasse *StahlBetonQuerschnittsbemessung*

Die Klasse *StahlBetonQuerschnittsbemessung* beinhaltet alle Bemessungsmethoden.

#### Attribute

```
int status;
int OK = 1;
int EINGABE_FEHLER = -1;
int KEINE_SCHUBBEWERTUNG_EFORDERLICH = -2;
double teta;
double tetaGrad;
```

**Methoden**

private double min(double a,double b)
Methode liefert den minimalen Wert der Attribute a und b.

private double max(double a,double b)
Methode liefert den maximalen Wert der Attribute a und b.

Konstruktor
public StahlBetonQuerschnittsbemessung()
public int getStatus()
Liefert den Status der Bemessung:

public double querkraftBemessung(VSd, d, fck, Asl, bw, sigmacd, alpha, z, xLast, roh_Beton, gammac, fyk, gammaS)

Diese Methode beinhaltet den gesamten Berechnungsablauf. Die Parameter lauten folgendermaßen:

```
VSd        // Bemessungsquerkraft [kN]
d          // statische Querschnittshöhe
fck        // Betonnenndruckfestigkeit [MN / m2]
Asl        // Querschnittsfläche der Längsbewehrung [cm2]
bw         // kleinste Querschnittsbreite [m]
sigmacd    // eingeprägte Betonspannungen aus Vorspannung oder Zwang [MN / m2]
alpha      // Neigung der Schubbewehrung [90° = senkrecht]
z          // innerer Hebelarm aus Biegezugbemessung i.A. z = 0.90 d  [m]
xLast      // Entfernung einer Einzellast vom betrachteten Schnitt [m]
roh_Beton  // Rohdichte eines Leichtbetons [kg / m3], für Normalbeton gilt roh_Beton = 1
gammac     // Sicherheitsbeiwert für Beton i.A. 1.5
fyk        // Stahlzugfestigkeit [MN / m2] für BSt IV = 500 MN / m2
gammaS     // Sicherheitsbeiwert für Betonstahl i.A. 1.15
```

Der Berechnungsablauf:

Ermittlung von $h_1$ in Abhängigkeit der Betonart

VRdct = widerstandsquerkraft(n1,Asi,fck,sigmacd,bw,d)

VRdmax = neigungsWinkel(alphaC, fck, bw, d, sigmacd, alpha,
gammac, n1, VSd, roh_Beton, z);

erf_asw = bemessungDerQuerkraft(VSd, fyk, gammaS, z, alpha)

private double widerstandsquerkraft(n1, Asl, fck, sigmacd, bw, d)
Ermittelung der Widerstandsquerkraft ohne Berücksichtigung einer Schubbewertung

```
private double neigungsWinkel(alphac, fck, bw, d, sigmacd, alpha, gammac,
                              n1, VSd, roh_Beton, z)
```
Bestimmung des Neigungswinkels der Betondruckstrebe

```
private double bemessungDerQuerkraft(VSd, fyk, gammaS, z, alpha)
```
Bemessungswert der Querkraft, die mit der eingelegten Schubbewertung aufnehmbar ist

```
private double cot(double winkel)
```
Bereitstellung der Funktion für den Kotangens eines Winkels.

## 2.6 Stahlbau

### 2.6.1 Theoretische Grundlagen

Alle hier aufgeführten Funktionen sind auf der Grundlage der DIN 18 800 (11.90) – Stahlbauten entstanden. Die Berechnungsformeln stammen zumeist aus [10]. Dabei werden die Trag-, Lage- und Gebrauchstauglichkeitsnachweise im Stahlbau grundsätzlich drei Nachweisverfahren unterschieden:

**Elastisch-Elastisch**

Beim Nachweisverfahren elastisch-elastisch sind sowohl die Beanspruchungen, als auch die Beanspruchbarkeiten nach der Elastizitätstheorie zu ermitteln.

**Elastisch-Plastisch**

Beim Nachweisverfahren elastisch-plastisch werden die Beanspruchungen nach der Elastizitätstheorie, bzw. nach dem Hook'schen Gesetz und die Beanspruchbarkeiten nach der Plastizitätstheorie ermitteln.

**Plastisch-Plastisch**

Beim Nachweisverfahren plastisch-plastisch sind sowohl die Beanspruchungen, als auch die Beanspruchbarkeiten nach der Plastizitätstheorie zu ermitteln. Grundgedanke hierbei ist, nicht nur den Querschnitt vollständig auszunutzen, sondern auch eventuelle Systemreserven, infolge von Lastumlagerungen auszunutzen. Das statische System wird somit bis zum rechnerischen Versagen beansprucht.

Weiterhin wird noch unterschieden, ob es sich um ein nicht stabilitätsgefährdetes Bauteil handelt oder nicht.

#### 2.6.1.1 Nachweis der Tragsicherheit

Die grundsätzliche Vorgehensweise ist die Ermittlung der Beanspruchung $S_d$ (engl.: stress). Hierbei handelt es sich um Designlasten, die mit einem Teilsicherheitsbeiwert $g$ multipliziert werden müssen. Bei den erstellten Nachweisfunktionen müssen bereits $g$-fache Lasten eingegeben werden. Die damit ermittelten Beanspruchungen müssen mit den aufnehmbaren Lasten $R_d$ (engl.: resistance) auf der Seite der Beanspruchbarkeiten verglichen werden.

Der Nachweis ist eingehalten, wenn gilt

$$S_d / R_d \leq 1 \qquad \text{Gl. (86)}$$

Im Rahmen des Tragsicherheitsnachweises müssen weiterhin die Nachweise auf Biegeknicken nach DIN 18800, Teil 2 (Abschnitt 4.1) und der Nachweis auf Biegedrillknicken nach DIN 18800 Teil 2 (Abschnitt 4.1) geführt werden.

**Abgrenzungskriterien für den Nachweis des Biegeknicken**

Der Nachweis erfolgt nach DIN 18800 Teil 2. Der Nachweis kann entfallen, wenn die maßgebenden Biegemomente nach Theorie II. Ordnung nicht größer sind als die 1,1-fachen Werte der Biegemomente nach Theorie I. Ordnung. Hiervon kann ausgegangen werden, wenn eine der folgenden Bedingungen eingehalten ist:

a)

$$\frac{N_d}{0.1 * N_{Ki,d}} \leq 1$$ Gl. (87)

mit

$$N_{Ki,d} = \frac{p^2 * (E * I)_d}{s_k{}^2}$$ Gl. (88)

$$(E * I)_d = \frac{E * I}{g_M}$$ Gl. (89)

$$s_k = b * l$$ Gl. (90)

b)

$$\frac{\bar{l}_k}{0.3 * \sqrt{\frac{f_{yd}}{s_{N,d}}}} \leq 1$$ Gl. (91)

mit

$$\bar{l}_k = \frac{l_k}{l_a}$$ Gl. (92)

$$l_k = \frac{s_k}{i}$$ Gl. (93)

$$l_a = p * \sqrt{\frac{E}{f_{y,k}}}$$ Gl. (94)

$$s_{N,d} = \frac{N_d}{A}$$ Gl. (95)

c)

$$b * e \leq 1$$ Gl. (96)

mit

$$e = l * \sqrt{\frac{N_d}{(E * I)_d}}$$ Gl. (97)

**Abgrenzungskriterien für den Nachweis des Biegedrillknicken**

Der Nachweis nach DIN 18800 Teil 2 darf entfallen,
- bei Stäben mit Hohlquerschnitt
- Wenn I-förmige Stäbe nur durch ein Biegemoment beansprucht werden
- wenn bei einfach symmetrischen Querschnitten (Symmetrie zur z-Achse), die durch ein Biegemoment $M_y$ beansprucht werden, deren Druckgurt im Abstand c seitlich unverschieblich gehalten ist und wenn gilt:

$$c \leq 0.5 * l_a * i_{z,g} * \frac{M_{pl,y,d}}{M_{y,d}}$$ Gl. (98)

### 2.6.1.2 Tragsicherheitsnachweis für nicht stabilitätsgefährdete Bauteile

#### 2.6.1.2.1 Elastisch-elastisch

· **Normalkraftbeanspruchung**

Für die Tragsicherheitsnachweise der Normalkraftbeanspruchung sind zwei Einzelnachweise zu unterscheiden:

- Druckkräfte
- Zugkräfte

Diese Nachweise sind bei der Programmierung für den Stahlbau in der Bibliothek *EdatraDLL.dll* enthalten. Allgemein werden für beide Beanspruchungsarten zunächst die Beanspruchungen $S_d$ , bzw. $N_d$ ermittelt.

$N_d$ als Designlast, welche den Stab in Normalrichtung beansprucht.

Dabei kann die Querschnittfläche theoretisch beliebige Form annehmen. Im Projekt Ed_tra hat man sich jedoch auf standardisierte I-Profile nach DIN 1025 beschränkt.

Weiterhin müssen die aufnehmbaren Lasten $N_{el|pl,d}$ auf Seite der Beanspruchbarkeiten $R_d$ (engl.: resistance) ermittelt werden.

Es wird nun nachgewiesen

$$\frac{N_d}{N_{el,d}} \leq 1$$ Gl. (99)

Die massgebenden Werte für *g* auf der Seite der Beanspruchungen, sowie die Werte für die Beanspruchbarkeiten können der DIN 18800 oder Werken wie [10] entnommen werden. Für die Werte $N_{el,d}$ gilt:

$$N_{el,d} = \frac{f_{yk}}{g_M} * A$$ Gl. (100)

mit

$f_{yk}$ Streckgrenze des verwendeten Stahls
$g_M$ Sicherheitsbeiwert für das Material
$A$ Querschnittsfläche

Durch Umstellung der Formeln können darüber hinaus ermittelt werden:

Für den Stahldruckstab:

- Nachweis auf Druck
- Berechnung der aufnehmbaren Last
- Berechnung des erforderlichen Profils

Für den Stahlzugstab:

- Nachweis auf Zug (führbar entweder über den Querschnitt oder das Profil)
- Berechnung des erforderlichen Querschnitts bzw. Profils
- Berechnung der aufnehmbaren Last (aufgrund des Profils oder des Querschnittes)

Bei nicht stabilitätsgefährdeten Bauteilen ist ein Nachweis auf Stabilität nicht zu führen. Die Abgrenzungskriterien wurden in Kapitel 2.6.1.1 genannt.

· **Biegung**

Der Nachweis auf Biegung beim Verfahren elastisch-elastisch wird über die Spannungen geführt. Der Nachweis ist in den Bemessungsfunktionen nicht enthalten Es wird nachgewiesen:

$$\frac{s_d}{s_{Rd}} \leq 1$$ Gl. (101)

mit

$$s_d = \left| \frac{N}{A} \pm \frac{M_y}{W_y} \pm \frac{M_z}{W_z} \right|$$ Gl. (102)

$$s_{Rd} = \frac{f_{yk}}{g_M} \quad \text{Gl. (103)}$$

| | |
|---|---|
| $s_{Rd}$ | Aufnehmbare Spannung des Querschnitts |
| $g_M$ | Sicherheitsbeiwert für das Material |
| $f_{yk}$ | Streckgrenze des verwendeten Stahls |
| $A$ | Querschnittsfläche |
| $My$ | Bemessungsmoment um die Y-Achse |
| $Mz$ | Bemessungsmoment um die Z-Achse |
| $Wy$ | Widerstandsmoment um die Y-Achse |
| $Wz$ | Widerstandsmoment um die Z-Achse |

· **Schubspannung**

Der Nachweis auf Schub beim Verfahren elastisch-elastisch wird wieder über die Spannungen geführt. Der Nachweis ist in den Bemessungsfunktionen nicht enthalten Es wird nachgewiesen:

$$\frac{t_d}{t_{Rd}} \leq 1 \quad \text{Gl. (104)}$$

mit

$$t_{Rd} = \frac{f_{yk}}{\sqrt{3} * g_M} \quad \text{Gl. (105)}$$

Die Schubspannung selbst ermittelt sich allgemein zu

$$t_d = \frac{V_z * S_y}{I_y * s} \quad \text{Gl. (106)}$$

oder vereinfacht bei doppeltsymmetrischen Querschnitten mit ausgeprägten Flanschen nach

$$t_d = \frac{V_z}{A_{Steg}} \quad \text{Gl. (107)}$$

mit

| | |
|---|---|
| $t_{Rd}$ | Aufnehmbare Schubspannung des Querschnitts |
| $g_M$ | Sicherheitsbeiwert für das Material |
| $f_{yk}$ | Streckgrenze des verwendeten Stahls |
| $A_{steg}$ | Stegfläche des I-Profils |
| $Vz$ | Bemessungsquerkraft in Z-Richtung |
| $Iy$ | Flächenträgheitsmoment II. Ordnung um die Y-Achse |
| $Sy$ | Flächenmoment I. Grades um die Y-Achse |
| $s$ | Stegstärke |

· **Vergleichsspannung**

Um den Spannungsnachweis zu erfüllen, muss weiterhin die Vergleichsspannung eingehalten werden.

$$\frac{s_{v,d}}{s_{Rd}} \leq 1 \qquad \text{Gl. (108)}$$

mit

$$s_{v,d} = \sqrt{s^{2} + 3 * t^{2}} \qquad \text{Gl. (109)}$$

Für nähere Informationen zu den Nachweisen für das Verfahren elastisch-elastisch wird auf die DIN 18800 und [10] verwiesen.

### 2.6.1.2.2 Elastisch plastisch

· **Spannungsnachweis**

Die Ermittlung der Beanspruchung erfolgt bei diesem Verfahren wiederum nach der Elastizitätstheorie. Weiterhin gelten für die Ermittlung der vollplastischen Grenzschnittgrößen die Annahmen, dass die linearelastisch-idealplastische Spannungs-Dehnungs-Beziehung gilt und die Querschnitte eben bleiben.

Die Grenzbiegemomente sind im allgemeinen für beliebige Querschnitte auf den 1.25 fachen Wert des Grenzbiegemoments im elastischen Zustand zu begrenzen. Dies gilt nicht für Einfeld- und Durchlaufträger mit konstantem Querschnitt. Füe gewalzte I-Profile, wie sie in den Funktionen programmiert wurden gilt demnach

$M_{pl,y,d} < 1.25 * s_{Rd} * W_y$ und $M_{pl,z,d} \geq 1.25 * s_{Rd} * W_z$

Die Grenzschnittgrößen ermitteln sich zu

$$N_{pl,d} = s_{Rd} * A \qquad \text{Gl. (110)}$$

$$M_{pl,y,d} = s_{Rd} * W_{pl,y} \qquad \text{Gl. (111)}$$

$$V_{pl,z,d} = t_{Rd} * (h-t) * s \qquad \text{Gl. (112)}$$

$$M_{pl,z,d} = s_{Rd} * W_{pl,z} \qquad \text{Gl. (113)}$$

$$V_{pl,y,d} = t_{Rd} * 2 * b * t \qquad \text{Gl. (114)}$$

Es ist nun nachzuweisen, dass die Grenzschnittgrößen im plastischen Zustand nicht überschritten sind.

**Vereinfachter Tragsicherheitsnachweis für doppeltsymmetrische I-Profile mit den Beanspruchungen $N$, $M_y$, $V_z$**

| Gültigkeitsbereich | $\frac{V_z}{V_{pl,z,d}} \leq 0.33$ | $0.33 < \frac{V_z}{V_{pl,z,d}} \leq 1$ |
|---|---|---|
| $\frac{N}{N_{pl,d}} \leq 0.1$ | $\frac{M_y}{M_{pl,y,d}} \leq 1$ | $0.88 * \frac{M_y}{M_{pl,y,d}} + 0.37 \frac{V_z}{V_{pl,z,d}} \leq 1$ |
| $0.1 < \frac{N}{N_{pl,d}} \leq 1$ | $0.9 * \frac{M_y}{M_{pl,y,d}} + \frac{N}{N_{pl,d}} \leq 1$ | $0.8 * \frac{M_y}{M_{pl,y,d}} + 0.89 \frac{N}{N_{pl,d}} + 0.33 \frac{V_z}{V_{pl,z,d}} \leq 1$ |

Tabelle 6. Vereinfachte Tragsicherheitsnachweise für doppeltsymmetrische I-Profile mit den Beanspruchungen $N$, $M_y$, $V_z$

**Vereinfachter Tragsicherheitsnachweis für doppeltsymmetrische I-Profile mit den Beanspruchungen $N$, $M_z$, $V_y$**

| Gültigkeitsbereich | $\frac{V_y}{V_{pl,y,d}} \leq 0.25$ | $0.25 < \frac{V_y}{V_{pl,y,d}} \leq 0.9$ |
|---|---|---|
| $\frac{N}{N_{pl,d}} \leq 0.3$ | $\frac{M_z}{M_{pl,z,d}} \leq 1$ | $0.95 * \frac{M_z}{M_{pl,z,d}} + 0.82 \left[\frac{V_y}{V_{pl,y,d}}\right]^2 \leq 1$ |
| $0.3 < \frac{N}{N_{pl,d}} \leq 1$ | $0.91 * \frac{M_z}{M_{pl,z,d}} + \left[\frac{N}{N_{pl,d}}\right]^2 \leq 1$ | $0.87 * \frac{M_z}{M_{pl,z,d}} + 0.95 * \left[\frac{N}{N_{pl,d}}\right]^2 + 0.75 * \left[\frac{V_y}{V_{pl,y,d}}\right]^2 \leq 1$ |

Tabelle 7. Vereinfachte Tragsicherheitsnachweise für doppeltsymmetrische I-Profile mit den Beanspruchungen $N$, $M_z$, $V_y$

**Tragsicherheitsnachweis für doppeltsymmetrische I-Profile mit den Beanspruchungen $N$, $M_y$, $V_z$, $M_z$, $V_y$**

Unter der Voraussetzung, dass die Schnittgrößen $V_z \leq 0.33 * V_{pl,z,d}$ und $V_y \leq 0.25 * V_{pl,y,d}$ sind darf der Tragsicherheitsnachweis mit den folgenden Formeln geführt werden

$$M_y^* = \left[1 - \left[\frac{N}{N_{pl,d}}\right]^{1.2}\right] * M_{pl,y,d}$$ Gl. (115)

Für $M_y \leq M^*_y$ gilt:

$$\frac{M_z}{M_{pl,z,d}} + c_1 + c_2 * \left[\frac{M_y}{M_{pl,y,d}}\right]^{2.3} \leq 1$$ Gl. (116)

mit

$$c_1 = \left[\frac{N}{N_{pl,d}}\right]^{2.6}$$ Gl. (117)

$$c_2 = (1 - c_1)^{-\frac{N_{pl,d}}{N}}$$ Gl. (118)

Für $M_y > M^*_y$ gilt:

$$\frac{1}{40} * \left[\frac{M_z}{M_{pl,z}} - \frac{M_z^*}{M_{pl,z,d}}\right] + \left[\frac{N}{N_{pl,d}}\right]^{1.2} + \frac{M_y}{M_{pl,y,d}} \leq 1$$ Gl. (119)

mit

$$\frac{M_z^*}{M_{pl,z,d}} = 1 - c1 - c2\left[\frac{M_y^*}{M_{pl,y,d}}\right]^{2.3}$$ Gl. (120)

Alle drei Nachweisarten sind in der Bibliothek *edtra_BdkJ.dll* inplementiert.

### 2.6.1.3 Tragsicherheitsnachweis für stabilitätsgefährdete Bauteile

**Biegeknicksicherheitsnachweis nach Theorie II Ordnung.**

· **Planmäßig mittiger Druck**

**Biegeknicken**

Der Berechnungsablauf ergibt sich wie folgt und wurde ebenfalls in der Bibliothek *EdatraDLL.dll* implementiert.

- Ermittlung der Knicklänge

$$s_k = b * l$$ Gl. (121)

*b* Knicklängenbeiwert
*l* Stablänge

- Schlankheitsgrad

- Schlankheitsgrad

$$l_k = s_k / l \qquad \text{Gl. (122)}$$

*i* Trägheitsradius

- Bezogener Schlankheitsgrad

$$\overline{l_k} = l_k / l_a \qquad \text{Gl. (123)}$$

mit dem Bezugsschlankheitsgrad

$$l_a = p * \sqrt{\frac{E}{f_{yk}}} \qquad \text{Gl. (124)}$$

- Einordnen des Querschnitts in eine Knickspannungslinie nach Tafel 8.35 aus [10].
- Abminderungsfaktor *k* nach den europäischen Knickspannungslinien

für $\overline{l_k} \leq 0.2$ gilt:

$$k = 1 \qquad \text{Gl. (125)}$$

für $\overline{l_k} > 0.2$ gilt:

$$k = \frac{1}{\left(k + \sqrt{k^2 - \overline{l}_k^2}\right)} \qquad \text{Gl. (126)}$$

$$k = 0.5 * \left[1 + a * \left(\overline{l_k} - 0.2\right) + \overline{l}_k^2\right] \qquad \text{Gl. (127)}$$

| Knickspannungslinie | a | b | c | d |
|---|---|---|---|---|
| α | 0.21 | 0.34 | 0.49 | 0.76 |

Tabelle 8. Parameter α zur Berechnung des Abminderungsfaktors κ

**Nachweis der Biegeknicksicherheit**

$$\frac{N_d}{\left(k * N_{pl,d}\right)} \leq 1 \qquad \text{Gl. (121)}$$

## Biegedrillknicken

Für Walzprofile mit I-förmigem Querschnitt, wie sie in den vorliegenden Berechnungsfunktionen zur Verwendung kommen, sowie für Hohlprofile ist bei Belastung durch mittigen Druck kein Biegedrillknicknachweis erforderlich. Für weiter Informationen sei daher auf [10] verwiesen.

· **Einachsige Biegung ohne Normalkraft**

Behandelt werden I-förmige Profile mit der Beanspruchung $M_y$. Bei einer Beanspruchung mit $M_z$ ist ein Biegedrillknicknachweis nicht erforderlich.

## Biegedrillknicken

Der Berechnungsablauf ergibt sich wie folgt und wurde in der Bibliothek *edtra_BdkJ.dll* programmiert.

- Bemessung des idealen Biegerdrillknickmoments $M_{ki,y,d}$

$$N_{Ki,y,d} = z * N_{Ki,z,d} * \left( \sqrt{c^2 + 0.25 * z_p^{\ 2}} + 0.5 * z_p \right) \qquad \text{Gl. (122)}$$

$N_{Ki,z,d}$ Bemessungswert der Normalkraft unter der kleinsten Verzweigungslast für das Ausweichen senkrecht zur z-Achse nach der Elastizitätstheorie.

$$N_{Ki,z,d} = \frac{p^2 * E * I_z}{l^2 * g_M} \qquad \text{Gl. (123)}$$

$g_M = 1.1$

*x* Momentenbeiwert für die Gabellagerung an den Stabenden

| Momentenverlauf | ξ |
|---|---|
| *maxM* | 1.00 |
| *maxM* | 1.12 |
| *maxM* | 1.35 |
| *maxM* $-1<\psi<1$ *y *maxM* | 1.77-0.77*ψ |

Tabelle 9. Momentenbeiwert ψ für die Gabellagerung an den Stabenden

*l* Abstand der Gabellager

*c* Drehradius des Querschnitts

$$c^2 = (I_w + 0.039 * l^2 * I_T) / I_z \qquad \text{Gl. (124)}$$

$I_w$ Wölbflächenmoment II. Grades bezogen auf den Schubmittelpunkt
$I_T$ Torsionsflächenmoment II. Grades
$z_p$ Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt, auf der Biegezugseite positiv($z_p$ > 0, wenn die Querlast am Zuggurt angreift)

Bei Trägerhöhen $h \leq 600$ mm darf $M_{Ki,y,d}$ wie folgt berechnet werden

$$N_{Ki,y,d} = \frac{1.32 * b * t * E * I_y}{l * h^2 \cdot g_M} \qquad \text{Gl. (125)}$$

- Bezogener Schlankheitsgrad

$$\overline{l_M} = \sqrt{\frac{M_{pl,y,d}}{M_{ki,y,d}}} \qquad \text{Gl. (126)}$$

- Abminderungsfaktor $k_M$ für die Biegemomente

Bei Trägerhöhen $h \leq$ 600mm, konstantem Querschnitt und l< b* t / h * 200*240/$f_{yk}$ gilt

$$k_M = 1 \; .$$

Für $\overline{l_M} \leq 0.4$ gilt

$$k_M = 1 \qquad \text{Gl. (127)}$$

Für $\overline{l_M} > 0.4$ gilt

$$k_M = \left[ \frac{1}{1 + \overline{l_M}^{2*n}} \right]^{1/n} \qquad \text{Gl. (128)}$$

$n$ Trägerbeiwert für das Biegedrillknicken. Für Walzprofile gilt $n$=2.5

**Nachweis der Biegedrillknicksicherheit**

$$\frac{M_{y,d}}{\left(k_M * M_{pl,y,d}\right)} \leq 1 \qquad \text{Gl. (129)}$$

· **Einachsige Biegung mit Normalkraft**

Der Berechnungsablauf ergibt sich wie folgt und wurde in die Bibliothek *edtra_BdkJ.dll* implementiert.

## Biegeknicken

- Ermittlung der Beanspruchungen $N_d$, und $M_d$
- Bezogener Schlankheitsgrad $\overline{l_K}$ nach Gl. (123)
- Abminderungsfaktor $k$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127).
- Momentenbeiwert $b_m$ für Biegeknicken nach Tabelle 10
- Wert $Dn$:

$$\Delta n = \frac{N_d}{k * N_{pl,d}} * \left[ \frac{1 - N_d}{k * N_{pl,d}} \right] * k^2 * \overline{l_K^2} \leq 0.1$$ Gl. (130)

## Nachweis der Biegeknicksicherheit

$$\frac{N_d}{k \cdot N_{pl,d}} + \frac{b_M * M_d}{M_{pl,d}} + \Delta n \leq 1$$ Gl. (131)

| Momentenverlauf | Momentenbeiwerte $b_m$ für Biegeknicken | Momentenbeiwerte $b_M$ für Biegedrillknicken |
|---|---|---|
| Stabendmomente<br>$M_1$ $y*M_1$<br>$-1 \leq y \leq 1$ | $b_{M,y}=0.66+0.44*y$<br>jedoch $b_{M,y} \geq 1-(1/h_{Ki,d}$[1]$)$<br>und $b_{M,y} \geq 0.44$ | $b_{M,y} = 1.8 - 0.7*y$ |
| Momente aus Querlast<br>$M_Q$ | $M_{m,Q}=1.0$ | $M_{M,Q}=1.3$ |
| $M_Q$ | | $M_{M,Q}=1.4$ |
| Momente aus Querlasten mit Stabendmomenten<br>$M_1$ $M_Q$ $DM$<br>$M_Q$ $M_1$ $DM$<br>$M_1$ $DM$ $M_Q$ | $\psi \leq 0.77$ : $b_m = 1.0$<br>$\psi > 0.77$ :<br>$b_m = \frac{M_Q + M_1 * b_{m,y}}{M_Q + M_1}$ | $b_M = b_{M,y} + \left(\frac{M_Q}{\Delta M}\right) * \left(b_{M,Q} - b_{M,y}\right)$<br>$M_Q$= \|maxM nur aus Querlast<br>Bei nicht durchschlagendem Momentenverlauf :<br>$\Delta M$ = \|maxM\|<br>Bei durchschlagendem Momentenverlauf :<br>$\Delta M$ = \|maxM\| + \|minM\| |
| [1] $h_{Ki,d}$ : Verzweigungslastfaktor des Systems. $h_{Ki,d}=N_{Ki,d}/N_d;=p^2*(E*I)_d/s_K^2$ | | |

Tabelle 10. Momentenbeiwert βm für Biegeknicken und Biegedrillknicken

## Biegedrillknicken

Der Berechnungsablauf ergibt sich wie folgt.

- Ermittlung der Beanspruchungen $N_d$, und $M_d$

- Bezogener Schlankheitsgrad $l_{K,z}$ für Ausweichen senkrecht zur Z-Achse

$$\overline{l_{K,z}} = \sqrt{\frac{N_{pl,d}}{N_{Ki,z,d}}} \qquad \text{Gl. (132)}$$

mit

$$N_{Ki,z,d} = \frac{p^2 * (E * I_z)_d}{s_{K,z}^{\;2}} \qquad \text{Gl. (133)}$$

$$(E * I_z)_d = \frac{E * I_z}{g_M} \qquad \text{Gl. (134)}$$

$s_{K,z}$ Knicklänge für das Ausweichen senkrecht zur Z-Achse

$$s_{K,z} = b_z * l \qquad \text{Gl. (135)}$$

- Abminderungsfaktor $k_z$ $(\overline{l_{K,z}})$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127).

- Abminderungsfaktor $k_M$ für das Biegedrillknicken nach Gl. (127) bis Gl. (128)

- Beiwert $k_y$ zur Berücksichtigung des Momentenverlaufs $M_y$ und des bezogenen Schlankheitsgrades $\overline{l}_{Kz}$ .

$$k_y = \frac{1 - N_d}{k_z * N_{pl,d}} * a_y \leq 1 \qquad \text{Gl. (136)}$$

$k_z$ Abminderungsfaktor nach den Europäischen Knickspannungslinien in Abhängigkeit vom bezogenen Schlankheitsgrad $\overline{l}_{Kz}$ , siehe oben.

$$a_y = 0.15 * \overline{l_{k,z}} * b_{M,y} - 0.15 \leq 0.9 \qquad \text{Gl. (137)}$$

$b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ nach Tabelle 10, Spalte 3

## Nachweis der Biegedrillknicksicherheit

$$\frac{N_d}{k_z \cdot N_{pl,d}} + \frac{M_{y,d}}{k_M M_{pl,y,d}} \cdot k_y \leq 1 \qquad \text{Gl. (138)}$$

· **Zweiachsige Biegung mit und ohne Normalkraft**

In DIN 18800 Teil 2 werden zwei Nachweismethoden für das Biegeknicken um zwei Achsen mit und ohne Normalkraft angeboten. Beide Nachweisarten sind in der Bibliothek *edtra_Bdk.dll* programmiert.

· **Biegeknicken nach Nachweismethode 1**

Der Berechnungsablauf ergibt sich wie folgt.

- Ermittlung der Beanspruchungen $N_d$, und $M_{y,d}$, $M_{z,d}$

- Bezogener Schlankheitsgrad $\overline{l}_K$ für das Ausweichen zur jeweiligen Achse nach Gl. (123)

$\overline{l}_{K,y}$ nach Gl. (132) bis Gl. (135)

$\overline{l}_{K,z}$ ergibt sich äquivalent

- Abminderungsfaktor $k_y$ bzw. $k_z$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127).

Beiwert $k_y$ bzw.$k_z$ zur Berücksichtigung des Momentenverlaufs $M_y$ bzw. $M_z$ und des bezogenen Schlankheitsgrades $\overline{l}_{Ky}$ bzw. $\overline{l}_{Kz}$ für die jeweilige Querschnittsachse

$$k_y = \frac{1 - N_d}{k_y * N_{pl,d}} * a_y \leq 1.5$$

Gl. (139)

$$a_y = \overline{l}_{k,y} \cdot 2 \cdot b_{M,y} - 4 + (a_{pl,y} - 1) \leq 0.8$$

Gl. (140)

$b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ nach Tabelle 10, Spalte 3

$a_{pl,y}$ plastischer Formbeiwert; $a_{pl,y}=1.14*W_{ply}/W_y$

$$k_z = \frac{1 - N_d}{k_z * N_{pl,d}} * a_z \leq 1.5$$

Gl. (141)

$$a_z = \overline{l}_{k,z} \cdot 2 \cdot b_{M,z} - 4 + (a_{pl,z} - 1) \leq 0.8$$

Gl. (142)

$b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ nach Tabelle 10, Spalte 3

$a_{pl,z}$ plastischer Formbeiwert; $a_{pl,z}=1.14*W_{plz}/W_z$

**Nachweis der Biegeknicksicherheit nach Methode 1**

$$\frac{N_d}{k \cdot N_{pl,d}} + \frac{M_{y,d}}{M_{pl,y,d}} \cdot k_y + \frac{M_{z,d}}{M_{pl,z,d}} \cdot k_z \leq 1$$

Gl. (143)

$k$ entspricht min ($k_y$, $k_z$)

· **Biegeknicken nach Nachweismethode 2**

In diesem Fall ist der Nachweis wie folgt zu führen

$$\frac{N_{Sd}}{k \cdot N_{pl,d}} + \frac{b_{m,y} \cdot M_{Sd,y}}{M_{pl,y,d}} \cdot k_y + \frac{b_{m,z} \cdot M_{Sd,z}}{M_{pl,z,d}} \cdot k_z + \Delta n \leq 1 \qquad \text{Gl. (144)}$$

$k$ entspricht min ($k_y$, $k_z$), Ermittlung siehe Nachweismethode 1

$b_{m.y}$, $b_{m,z,}$ Momentenbeiwert für Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ und $M_{z,d}$. nach Tabelle 10, Spalte 2

Beiwerte $k$:

$k_y < k_z$ → $k_y$=1 und $k_z$=$c_z$
$k_y = k_z$ → $k_y$=1 und $k_z$=1
$k_y < k_z$ → $k_y$=$c_y$ und $k_z$=1
mit

$$c_z = \frac{1}{c_y} = \frac{1 - \left( N_d / N_{pl,d} \right) * \overline{l_{K,y}}^2}{1 - \left( N_d / N_{pl,d} \right) * \overline{l_{K,z}}^2} \qquad \text{Gl. (145)}$$

und

$$c_y = \frac{1}{c_z} \qquad \text{Gl. (146)}$$

· **Biegedrillknicken nach Nachweismethode 2**

Der Nachweis ist wie folgt zu führen

$$\frac{N_d}{k_z \cdot N_{pl,d}} + \frac{M_{y,d}}{k_M * M_{pl,y,d}} \cdot k_y + \frac{M_{z,d}}{M_{pl,z,d}} \cdot k_z \leq 1 \qquad \text{Gl. (147)}$$

$k_z$ Abminderungsfaktor für das Biegeknicken nach den Europäischen Knickspannungslinien in Abhängigkeit vom bezogenen Schlankheitsgrad $\overline{l}_{Kz}$ nach Gl. (125) bis Gl. (127).

$k_M$ Abminderungsfaktor für das Biegedrillknicken nach Gl. (127) bis Gl. (128).

$k_y$ Beiwert zur Berücksichtigung des Verlaufs von $M_y$ nach Gl. (136) bis Gl. (137)

$k_z$ Beiwert zur Berücksichtigung des Verlaufs von $M_z$ nach

### 2.6.2 Programmierung in Java

Für die Nachweise des Stahlbaus entstanden zwei Bibliotheken. Die Bibliothek *Edatra.dll*, enthält die Nachweise für auf Druck und Zug beanspruchte Stäbe. Für zugbeanspruchte Stäbe sind dies die Nachweise nach den Verfahren elastisch-elastisch, sowie elastisch-plastisch für nicht stabilitätsgefährdete Bauteile. Für auf Druck beanspruchte Stäbe enthält die Bibliothek den Nachweise nach Theorie II. Ordnung für das Biegeknicken bei planmäßig mittigem Druck.

Weiterhin wurde die Bibliothek *edtra_BDKJ.dll* programmiert, sie enthält die Spannungsnachweise nach dem Verfahren elastisch-plastisch für nicht stabilitätsgefährdete Bauteile. Weiterhin enthält sie Stabilitätsnachweise für das Biegeknicken und Biegedrillknicken nach Theorie II. Ordnung für einachsige und zweiachsige Biegung mit und ohne Normalkraft. Für zweiachsige Biegung wurden die Stabilitätsnachweise sowohl nach Methode I, sowie auch nach Methode II implementiert.

Eine weitere Bibliothek, edtra_PrfJ.dll, enthält Funktionen mit der Querschnittswerte genormter I-förmiger Walzprofile ermittelt werden können.

#### 2.6.2.1 Die Bibliothek *EdatraDLL.dll*

Die Bibliothek *edatraDLL.dll* beinhaltet die Berechnungen für die Nachweise von Holz- und Stahlbauteilen für zug- und druckbeanspruchte Querschnitte. In Bild 82 sind die Klassen für die Nachweise für den Holzbau der Übersichtlichkeit halber grau hinterlegt worden. Diese werden im Kapitel Holzbau erläutert.

Die Klassen lassen sich unterscheiden in Klassen, welche exportiert werden und anschließend als COM Objekt verwendet werden können, und Klassen, welche nicht exportiert werden und nur innerhalb der Klassenhierarchie verwendet werden können.

Folgende Klassen werden nur intern verwendet:

- FunktionsObjekt.java
- Liste.java
- Liste2.java

Zu den exportierten Klassen zählen:

- StahlDruckStab.java ( COM Objekt → StahlDruckStab )
- StahlZugStab.java ( COM Objekt → StahlZugStab )

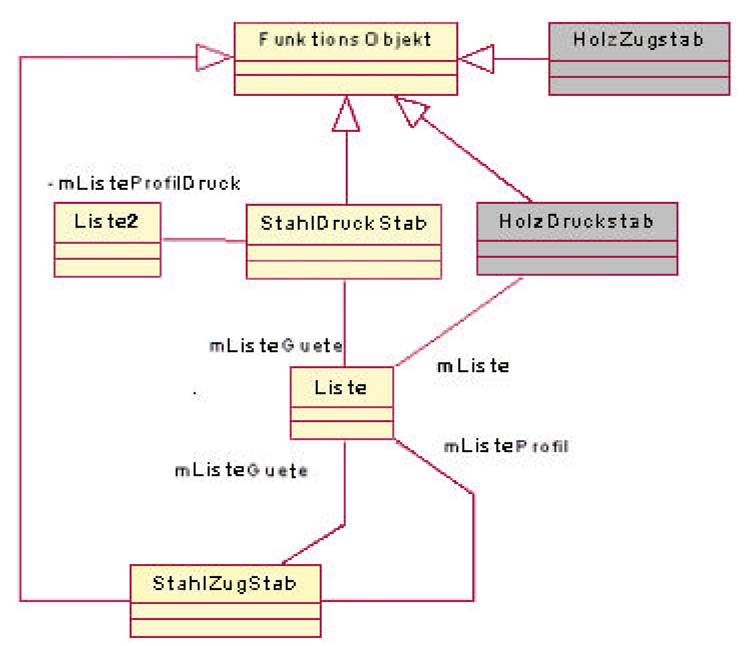



Bild 82. UML Klassendiagramm von *EdatraDLL.dll*

## Die Klasse *FunktionsObjekt*

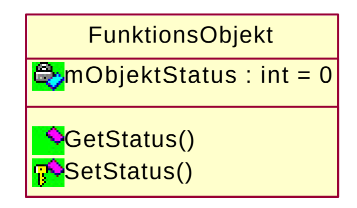



Bild 83. UML Darstellung der Klasse *FunktionsObjekt*

### Attribute

int mObjektStatus

*mObjektStatus* ist eine *private* Variable die den Status des Objekts angibt, d.h. beispielsweise, ob eine Berechnung erfolgreich durchgeführt wurde oder nicht. Da die Variable in einer übergeordneten Klasse deklariert wurde steht *mObjektStatus* in allen abgeleiteten Klassen zur Verfügung.

### Methoden

public int GetStatus ()

Die Methode dient zum Auslesen des Status des Objekts

protected int SetStatus (int status)

Die Methode dient zum Verändern des Status des Objekts

## Die Klasse *Liste*

Die Klasse *Liste* dient zum Einlesen einer Datei. Die abgespeicherten Werte können dann gezielt ausgelesen werden.

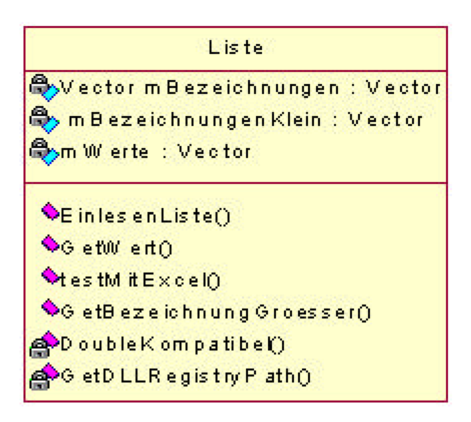



Bild 84. UML Darstellung der Klasse *Liste*

## Attribute

private Vector mBezeichnungen;
Im Vector *mBezeichnungen* werden die in der Datei hinterlegten Zeichenketten abgelegt. Das Format entspricht dem wie in der Datei (Groß- und Kleinschreibung), Leerzeichen am Anfang oder Ende werden entfernt.

private Vector mBezeichnungenKlein;
Im Vector *mBezeichnungenKlein* werden die in der Datei hinterlegten Zeichen abgelegt. Die in der Datei hinterlegte Zeichenkette wird in Kleinbuchstaben umgewandelt, Leerzeichen am Anfang oder Ende werden entfernt.

private Vector mWerte;
Im Vector *mWerte* wird die Ziffernfolge aus der Datei als Datentyp Double abgelegt. Die Werte können den Bezeichnern über den Index zugeordnet werden

## Methoden

public void EinlesenListe(String dateiName)
Die Methode liest die Datei mit dem *dateiName* ein. Im Parameter *dateiName* muss der Dateiname mit dem absoluten Pfad angegeben werden. Das Zeichen "\" muß als "\\" übergeben werden, da es sonst als ESC-Zeichen interpretiert wird. Bsp.: "C:\\temp\\Stahlguete.txt". Bei den Dateien handelt es sich um eigene geschriebene Dateien, welche Werte für Profile oder beispielsweise Holzgüten enthalten. Der Aufbau der Datei wird im Klassenkopfkommentar im Quellcode beschrieben.

public double GetWert(String bezeichnung)
Die Methode liefert den Wert der zu der im String *bezeichnung* übergebenen Zeichenkette gehört Groß- und Kleinschreibung wird beim Vergleich nicht berücksichtigt.

public String GetBezeichnungKleiner(double wert)
Die Methode gibt die nächst kleinere Bezeichnung zurück, d.h. der Tabellenwert der zurückgelieferten Bezeichnung ist der nächstkleinere in Bezug auf den übergebene Wert.
public String GetBezeichnungGroesser(double wert)
Die Methode gibt die nächst höhere Bezeichnung zurück, d.h. der Tabellenwert der zurückgelieferten Bezeichnung ist der nächst größere in Bezug auf den übergebenen Wert

private boolean DoubleKompatibel(String testString)
Die Methode überprüft, ob die durch *testString* übergebene Zeichenkette in einen gültigen Doublewert umgewandelt werden kann, d.h. es darf max. ein '.' und sonst nur Zahlen als Zeichen vorkommen. Kann der String in ein Doublewert umgewandelt werden, wird *true* zurückgeliefert, ansonsten *false*.

private String GetDLLRegistryPath()
Diese Methode liest den Pfad aus der Windows-Registry unter dem die DLL installiert wurde. Das macht das Lesen der Tabellendateien unabhängig vom Ort der Installation. Der aktuelle Schlüssel lautet // 04A335C6-EAD9-11D4-8AC0-00104B471BFE.

### Die Klasse *Liste2*

Die Klasse *Liste2* ist vom Aufbau her identisch der Klasse *Liste*. Es wurden lediglich sechs anstatt einer Variablen vom Typ Vector deklariert. Der Unterschied der beiden Klassen besteht demnach in der Anzahl der Werte, die aus einer Datei ausgelesen werden. So kann an dieser Stelle auf die Klasse Liste verwiesen werden.

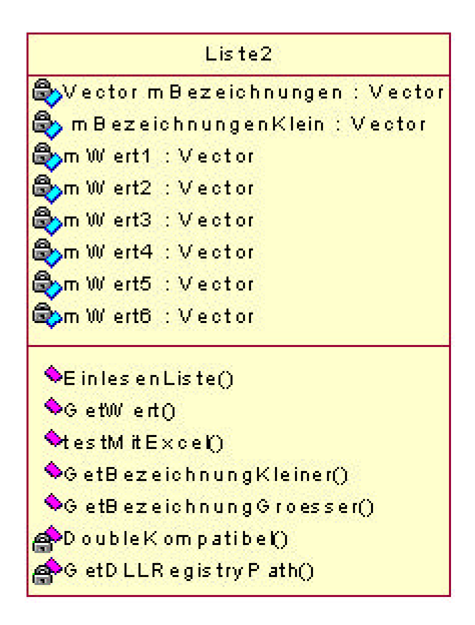



Bild 85. UML Darstellung der Klasse *Liste2*

### Die Klasse *StahlZugStab*

Die Klasse stellt verschiedene Funktionen für Nachweise für auf Zug beanspruchte Stahlquerschnitte zur Verfügung.

### Attribute

private String mDateiName
In *mDateiname* wird der vollständige Pfad der zu lesenden Datei gespeichert.

private double mStreckGrenze
Enthält die zur Stahlguete gehörende Streckgrenze

private String mStahlGuete
Enthält einen String, welcher mit "St" beginnt und an den die übergebene Stahlguete des Profils angehängt wird.
private String mProfil
Variable für die Profil Nennhöhe des Querschnitts

// Variablen für die Bemessungsmethoden
private int mProfilHoehe
Variable für die Profilhöhe des Querschnitts

private double mQuerschnitt;
Variable für den erforderlichen Querschnitt des Profils.

private Liste mListeGuete
Objektvariable vom Typ Liste, in dem in einem Vector die Stahlgüte gespeichert ist.

private Liste mListeProfil
Objektvariable vom Typ Liste, in dem in einem Vector die Daten des Profils gespeichert sind.

// Variablen für Belastungs Methoden
private double mAufnehmbareLast;
Variable für die ermittelte aufnehmbare Last. Wird in Methoden verwendet, in denen die aufnehmbare Belastung ermittelt werden soll.

**Methoden**

public double BemessungQuerschnitt(double beanspruchung, String stahlGuete)
Die Methode ermittelt anhand der übergebenen Beanspruchung und Stahlgüte den erforderlichen Querschnitt und liefert diesen zurück. Die Beanspruchung wird übergeben in kN = 1000 N, der Querschnitt wird in cm² zurückgeliefert. Bei der Variablen *stahlGuete* wird die Groß- bzw. Kleinschreibung, führende und abschließende Leerzeichen nicht beachtet.

public double BemessungQuerschnittV2(double beanspruchung, int stahlGuete)
Die Methode ermittelt wie die Methode *BemessungQuerschnitt* anhand der übergebenen Beanspruchung und *stahlGuete* (als int Wert!!) den erforderlichen Querschnitt und liefert diesen zurück. Die Beanspruchung wird ebenfalls übergeben in kN = 1000 N und der Querschnitt wird auch in cm² zurückgeliefert. Als *stahlGuete* wird lediglich die Ziffernfolge übergeben und nicht die gesamte Bezeichnung, z.B. St52 als 52.



Bild 86. UML Darstellung der Klasse *StahlZugStab*

public String BemessungProfil(double beanspruchung, String stahlGuete, String profilReihe)
Die Methode ermittelt anhand der übergebenen Beanspruchung und Stahlgüte das erforderliche Profil der Reihe *profilReihe* und liefert die erforderliche Nennhöhe als Stringvariable zurück. Die Profilreihen, welche unterstützt werden hängt lediglich davon ab, ob eine Textdatei existiert, in welchem die Daten hinterlegt sind (Datei Profil_<profilReihe>.txt). Defaultmässig werden die Profilreihen I, IPE, HEA, HEB und HEM bereitgestellt. Erweiterungen können durch den Benutzer selbst vorgenommen werden. Genauere Informationen zur Datei stehen im Klassenkopfkommentar der Klasse Liste. Bei der Übergabe der Werte *profilReihe* und *stahlGuete* wird die Groß- und Kleinschreibung nicht beachtet. Ebenso nicht beachtet werden Leerzeichen vor oder nach der eigentlichen Bezeichnung.

public int BemessungProfilI(double beanspruchung, int stahlGuete)
Die Methode ermittelt anhand der übergebenen Beanspruchung und Stahlgüte die erforderliche Profilhöhe für ein Profil der I-Reihe. Zurückgeliefert wird die Profilhöhe als Integer Wert, z.B. wird bei einem Profil I450 nur 450 als Integer zurückgegeben. Als *stahlGuete* wird die Ziffernfolge übergeben, also nicht die gesamte Bezeichnung, z.B. für St52, nur 52.

Die folgenden vier Methoden haben denselben strukturellen Aufbau und dieselben Rückgabewerte und sind nur für andere Profilreihen zuständig. So wird auf eine genauere Beschreibung an dieser Stelle verzichtet.

- public int BemessungProfilIPE(double beanspruchung, int stahlGuete)
- public int BemessungProfilHEA(double beanspruchung, int stahlGuete)
- public int BemessungProfilHEB(double beanspruchung, int stahlGuete)
- public int BemessungProfilHEM(double beanspruchung, int stahlGuete)

public double BelastungUeberQuerschnitt(double querschnitt, String stahlGuete)
Die Methode berechnet die aufnehmbare Last des Querschnitts anhand der Übergabewerte *querschnitt* als double Wert in cm² und s*tahlGuete* als String. Die Groß-, bzw. Kleinschreibung, führende Leerzeichen und abschließende Leerzeichen werden ignoriert. Die aufnehmbare Last wird in kN zurückgegeben.

public double BelastungUeberQuerschnittV2(double querschnitt, int stahlGuete)
Die Funktion berechnet anhand der Übergabewerte *querschnitt* als double Wert in cm2 und *stahlGuete* hier als int Wert die aufnehmbare Last des Querschnitts. Die aufnehmbare Last wird in kN zurückgegeben. Anstatt der Gesamtbezeichnung der Stahlgüte, wird in der Variable *stahlGuete* lediglich der Ziffernteil übergeben, also z.B. St52 anstatt 52.

public double BelastungUeberProfil(String profil, String stahlGuete)
Die Funktion berechnet anhand der Übergabewerte *profil* (String) und *stahlGuete* (String) die aufnehmbare Last des Querschnitts. Die Groß-, bzw. Kleinschreibung, führende Leerzeichen

und abschließende Leerzeichen werden bei beiden Werten ignoriert. Die aufnehmbare Last wird in kN zurückgegeben.

public double BelastungUeberProfilI(int profilNennHoehe, int stahlGuete)

Die Methode berechnet anhand der Übergabewerte *profil* (int) und stahlGuete (int) die aufnehmbare Last. Die aufnehmbare Last wird in kN zurückgegeben. Bei *profil* und *stahlGuete* wird jeweils nur der Ziffernteil Übergeben und nicht die gesamte Bezeichnung,lso z. B. bei Stahlgüte St52 nur 52 oder bei Profil HEB360 nur 360.

Die folgenden vier Methoden haben denselben strukturellen Aufbau und dieselben Rückgabewerte und sind nur für andere Profilreihen zuständig. So wird auf eine genauere Beschreibung an dieser Stelle verzichtet.

- public double BelastungUeberProfilIPE(int profilNennHoehe, int stahlGuete)
- public double BelastungUeberProfilHEA(int profilNennHoehe, int stahlGuete)
- public double BelastungUeberProfilHEB(int profilNennHoehe, int stahlGuete)
- public double BelastungUeberProfilHEM(int profilNennHoehe, int stahlGuete)

public double NachweisUeberQuerschnitt(double beanspruchung, double querschnitt, String stahlGuete)

Die Funktion führt den Nachweis auf Zug anhand der übergebenen Werte *beanspruchung* (double, in kN), *querschnitt* (double, in cm2) und *stahlGuete* (String). Bei *stahlGuete* wird die Groß-und Kleinschreibung, führende und abschließende Leerzeichen nicht beachtet. Als Rückgabewert wird der errechnete Wert geliefert. Hierbei handelt es sich um den Verhältniswert von angreifender Last zu aufnehmbarer Last. Der Wert ist dimensionslos.

public double NachweisUeberQuerschnittV2(double beanspruchung, double querschnitt, int stahlGuete)

Die Methode führt analog der Methode *NachweisUeberQuerschnitt* den Nachweis auf Zugbeanspruchte Querschnitte, mit dem Unterschied, dass die Stahlgüte als int Wert übergeben wird, z.B. für St52 nur 52. Als Rückgabewert wird der errechnete Wert geliefert. Hierbei handelt es sich wieder um den dimensionslosen Verhältniswert von angreifender Last zu aufnehmbarer Last.

public double NachweisUeberProfil(double beanspruchung, String profil, String stahlGuete)

Die Methode führt analog der Methode *NachweisUeberQuerschnitt* den Nachweis auf Zugbeanspruchte Querschnitte,mit dem Unterschied, dass anstatt eines Querschnitts eine Profilbezeichnung (z.B. HEB360) übergeben wird. Anhand dieser Profilbezeichnung wird dann der notwendige Querschnitt ermittelt, der die Last aufnehmen kann. Bei *stahlGuete* und *profil* werden die Groß-und Kleinschreibung, führende und abschließende Leerzeichen nicht beachtet. Der Rückgabewert entspricht ebenfalls der Methode *NachweisUeberQuerschnitt*.

public double NachweisUeberProfilI(double beanspruchung, int profilNennHoehe, int stahlGuete)

Die Methode liefert analog der Methode *NachweisUeberProfil* den Nachweis auf Zugbeanspruchte Querschnitte, mit dem Unterschied, dass *stahlGuete* sls int Wert übergeben wird, ebenso wird anstatt der kompletten Profilbezeichnung lediglich die *profilNennHoehe* als int Wert übergeben, z. B. HEB360 mit 360 oder die Stahlgüte St37 als 37.

Die folgenden vier Methoden haben denselben strukturellen Aufbau und dieselben Rückgabewerte und sind nur für andere Profilreihen zuständig. So wird auf eine genauere Beschreibung an dieser Stelle verzichtet.

- public double NachweisUeberProfilIPE(double beanspruchung, int profilNennHoehe, int stahlGuete)
- public double NachweisUeberProfilHEA(double beanspruchung, int profilNennHoehe, int stahlGuete)
- public double NachweisUeberProfilHEB(double beanspruchung, int profilNennHoehe, int stahlGuete)
- public double NachweisUeberProfilHEM(double beanspruchung, int profilNennHoehe, int stahlGuete)

private void SetStahlGuete(int stahlGuete)
Die Funktion bildet einen String, welcher mit "St" beginnt und an den die übergebene *stahlGuete* angehängt wird. Dieser String wird dann in die Variable *mStahlGuete* geschrieben.

private void SetProfil(String profilReihe, int profilNennHoehe)
Die Funktion bildet einen String, welcher der Profilreihe (*profilReihe*) entspricht und an den die *profilNennHoehe* angehängt wird. Dieser String wird dann in die Variable *mProfil* geschrieben.

private void UeberpruefeGueltigkeit(double beanspruchung)
Die Methode überprüft ob der Übergebene Wert gültig oder brauchbar ist. Also, ob beispielsweise die übergebene Beanspruchung $\geq$ 0 ist. Ist der übergebene Wert negativ wird mObjectStatus auf -1 gesetzt, sonst auf 0.

private void ResetDateiName()
Die Methode setzt den Pfad in *mDateiName* auf "" und löscht somit vorherige Angaben.

private void SetProfilHoehe(int anzProfilReihe)
Die Methode bestimmt anhand des Eintrags in *mProfil* die ProfilHoehe und speichert diese im Attribut *mProfilHoehe* ab. In *anzProfilReihe* wird die Anzahl der Buchstaben übergeben, welche in *mProfil* als Profilreihenbezeichnung abgelegt ist.

private void FuelleListeGuete()
Die Methode liest die Daten aus der Datei *Stahlguete.txt* in die Liste *mListeGuete* ein.

private void FuelleListeProfil(String profilReihe)
Die Methode liest die Daten aus der Datei Profil_<profilReihe>.txt in die Liste *mListeProfil* ein.

private String FormatiereFormat1(String zeichenKette)
Die Methode entfernt die Leerzeichen am Anfang und am Ende des Strings *zeichenKette*, zusätzlich werden alle Buchstaben in Großbuchstaben umgewandelt.

private int GetIndexZiffer(String zeichenKette)
Die Methode sucht das erste Vorkommen einer Ziffer, '1' .bis.'9' im String *zeichenKette.* Der Index der Stelle wird zurückgegeben. Die Zaehlung beginnt mit 0!

private boolean IntegerKompatibel(String testString)

Die Methode überprüft, ob die durch *testString* übergebene Zeichenkette in einen gültigen Integer Wert umgewandelt werden kann. Kann der String in ein Integer Wert umgewandelt werden, wird *true* zurückgeliefert, ansonsten *false.*

## Die Klasse *StahlDruckStab*

| StahlDruckStab |
|---|
| mAbminderungsFaktor : double |
| mBezogenerSchlankheitsGrad : double |
| mSchlankheitsGradY : double |
| mSchlankheitsGradZ : double |
| mProfil : String |
| mProfilHoehe : int |
| mQuerschnitt : double |
| mHoehe : type = double |
| mBreite : double |
| mTg : double |
| mIz : double |
| mIy : double |
| mKnickLaengeZ : double |
| mKnickLaengeY : double |
| mDateiName : StringI |
| mStreckGrenze : double |
| mStahlGuete : String |
| mListeGuete : Liste |
| mListeProfilDruck : Liste2 |
| Belastung() |
| BelastungI() |
| BelastungIPE() |
| BelastungHEA() |
| BelastungHEB() |
| BelastungHEM() |
| Nachweis() |
| NachweisI() |
| NachweisIPE() |
| NachweisHEA() |
| NachweisHEB() |
| NachweisHEM() |
| Bemessung() |
| BemessungI() |
| BemessungIPE() |
| BemessungHEA() |
| BemessungHEB() |
| BemessungHEM() |
| SetStahlGuete() |
| SetProfil() |
| UeberpruefeGueltigkeit() |
| ResetDateiName() |
| SetProfilHoehe() |
| FuelleListeGuete() |
| FuelleListeProfilDruck() |
| FormatiereFormat1() |
| GetIndexZiffer() |
| IntegerKompatibel() |
| SetzeProfilWerte() |
| BerechneSchlankheitsGrade() |
| BerechneAbminderungsFaktor() |

### Attribute

private double mAbminderungsFaktor;
// Variable für den Abminderungsfaktor Kappa für den Biegeknicknachweis

private double mBezogenerSchlankheitsGrad;
// Variable für den bezogenen Schlankheitsgrad $\lambda_{K_quer}$

private double mSchlankheitsGradY;
Variable für den Schlankheitsgrad des Stabes für das Biegeknicken um die Y-Achse

private double mSchlankheitsGradZ;
Variable für den Schlankheitsgrad des Stabes für das Biegeknicken um die Z-Achse

// Für mehrere Methoden
private String mProfil
Variable für die Profil Nennhöhe des Querschnitts

private int mProfilHoehe
Variable für die Profilhöhe des Querschnitts

private double mQuerschnitt // Profilquerschnitt

private double mHoehe // Profilhöhe

private double mBreite // Profilbreite

private double mTg // Flanschdicke des Profils

private double mIz // Trägheitsradius für Z-Achse

private double mIy // Trägheitsradius für Y-Achse

private double mKnickLaengeZ
// Knicklänge für das Knicken um die Z-Achse

private double mKnickLaengeY
// Knicklänge für das Knicken um die Z-Achse

Bild 87. UML Darstellung der Klasse *StahlDruckStab*

private String mDateiName
In *mDateiname* wird der vollständige Pfad der zu lesenden Datei gespeichert. enthält den vollst

private double mStreckGrenze
Enthält die zur Stahlguete gehörende Streckgrenze
private String mStahlGuete = new String();

private String mStahlGuete
Enthält einen String, welcher mit "St" beginnt und an den die übergebene Stahlgüte des Profils angehängt wird.

private Liste mListeGuete
Objektvariable vom Typ Liste, in dem in einem Vector die Stahlgüte gespeichert ist.

private Liste mListeProfilDruck
Objektvariable vom Typ *Liste2*, in dem in einem Vector die Daten des Profils gespeichert sind.

**Methoden**

public double Belastung(String profil, String stahlGuete, double knickLaengeY, double knickLaengeZ)
Die Methode berechnet anhand der Übergabewerte *profil* (String), *stahlGuete* (String), *knickLaengeY* (double, in cm) und *knickLaengeZ* (double, in cm) die aufnehmbare Last des Stabes. Groß- und Kleinschreibung, führende Leerzeichen und abschließende Leerzeichen werden bei den String Werten ignoriert. Die aufnehmbare Last wird in kN zurückgegeben

Grundlage für die Ermittlung der Knickspannungslinie und somit dem Abminderungsfaktor ist die Tabelle auf Seite 8.36 in [1].

public double BelastungI(int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)
Die Methode berechnet anhand der Übergabewerte *profilNennHoehe* (int), *stahlGuete* (int), *knickLaengeY* (double, in cm) und *KnickLaengeZ* (double, in cm) die aufnehmbare Last eines Stabes. Die Groß- und Kleinschreibung, führende Leerzeichen und abschließende Leerzeichen werden bei den String-Werten ignoriert. Die aufnehmbare Last wird in kN zurückgegeben. Bei *profil* und *stahlGuete* wird jeweils nur der Ziffernteil übergeben und nicht die gesamte Bezeichnung, z. B. Stahlguete St52 als 52 oder Profil HEB360 als 360.

Die folgenden vier Methoden haben denselben strukturellen Aufbau und dieselben Rückgabewerte und sind nur für andere Profilreihen zuständig. So wird auf eine genauere Beschreibung an dieser Stelle verzichtet.

- public double BelastungIPE(int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)
- public double BelastungHEA(int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)

- public double BelastungHEB(int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)
- public double BelastungHEM(int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)

public double Nachweis(double beanspruchung, String profil, String stahlGuete, double knickLaengeY, double knickLaengeZ)

Die Methode führt den Nachweis auf Druck anhand der übergebenen Werte *beanspruchung* (double, in kN), *profil* (String), *stahlGuete* (String), *knickLaengeY* (double in cm) und *knickLaengeZ* (double in cm). Bei stahlGuete und profil wird die Groß- und Kleinschreibung, führende und abschließende Leerzeichen nicht beachtet.

Grundlage für die Ermittlung der Knickspannungslinie und somit dem Abminderungsfaktor ist die Tabelle auf Seite 8.36 in [1]

public double NachweisI(double beanspruchung, int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)

Die Methode führt den Nachweis auf Druck anhand der übergebenen Werte *beanspruchung* (double, in kN), *profilNennHoehe* (int), *stahlGuete* (int), *pKnickLaengeY* (double in cm) und *pKnickLaengeZ* (double in cm). Bei *profilNennHoehe* und *stahlGuete* wird jeweils nur der Ziffernteil übergeben und nicht die gesamte Bezeichnung, z. B. Stahlguete St52 als 52 oder Profil HEB360 als 360.

Grundlage für die Ermittlung der Knickspannungslinie und somit dem Abminderungsfaktor ist wieder die Tabelle auf Seite 8.36 in [1].

Die folgenden vier Methoden haben denselben strukturellen Aufbau und dieselben Rückgabewerte und sind nur für andere Profilreihen zuständig. So wird auf eine genauere Beschreibung an dieser Stelle verzichtet.

- public double NachweisIPE(double beanspruchung, int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)
- public double NachweisHEA(double beanspruchung, int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)
- public double NachweisHEB(double beanspruchung, int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)
- public double NachweisHEM(double beanspruchung, int profilNennHoehe, int stahlGuete, double pKnickLaengeY, double pKnickLaengeZ)

public String Bemessung(double beanspruchung, double knickLaengeY, double knickLaengeZ, String stahlGuete, String profilReihe)

Die Methode ermittelt anhand der Übergabewerte *beanspruchung* (double in kN), *stablaenge* (double in cm), *knickLaengeY* (double in cm), *knickLaengeZ* (double in cm), *stahlGuete* (String) und *profilReihe* (String) das erforderliche Profil der Profilreihe *profilReihe*. Bei der Übergabe der Werte *profilReihe* und *stahlGuete* wird die Groß- und Kleinschreibung nicht beachtet. Ebenso nicht beachtet werden Leerzeichen vor oder nach der eigentlichen Bezeichnung.

public int BemessungI(double beanspruchung, double pKnickLaengeY, double pKnickLaengeZ, int stahlGuete)

Die Methode ermittelt anhand der Übergabewerte *beanspruchung* (double in kN), *knicklaengeY* (double in cm), *knickLaengeZ* (double in cm) und der *stahlGuete* (int) das erforderliche Profil der Profilreihe. Die erforderliche Profilnennhöhe wird im Erfolgsfall zurückgegeben

Die folgenden vier Methoden haben denselben strukturellen Aufbau und dieselben Rückgabewerte und sind nur für andere Profilreihen zuständig. So wird auf eine genauere Beschreibung an dieser Stelle verzichtet.

- public int BemessungIPE(double beanspruchung, double pKnickLaengeY, double pKnickLaengeZ, int stahlGuete)
- public int BemessungHEA(double beanspruchung, double pKnickLaengeY, double pKnickLaengeZ, int stahlGuete)
- public int BemessungHEB(double beanspruchung, double pKnickLaengeY, double pKnickLaengeZ, int stahlGuete)
- public int BemessungHEM(double beanspruchung, double pKnickLaengeY, double pKnickLaengeZ, int stahlGuete)

private void SetStahlGuete(int stahlGuete)

Die Methode bildet einen String, welcher mit "St" beginnt und an den die übergebene stahlGuete angehängt wird. Dieser String wird dann in die Variable *mStahlGuete* geschrieben

private void SetProfil(String profilReihe, int profilNennHoehe)

Die Methode bildet einen String, welcher der *profilReihe* beginnt und an den die *profilNennHoehe* angehängt wird. Dieser String wird dann in die Variable *mProfil* geschrieben.

private void UeberpruefeGueltigkeit(double beanspruchung)

Die Methode überprüft ob der übergebene Wert gültig oder brauchbar ist. Also, ob beispielsweise die übergebene Beanspruchung $\geq 0$ ist. Ist der übergebene Wert negativ, wird mObjectStatus auf -1 gesetzt, sonst auf 0.

private void ResetDateiName()

Die Methode setzt den Pfad in *mDateiName* auf "" und löscht somit vorherige Angaben.

private void SetProfilHoehe(int anzProfilReihe)
Die Methode bestimmt anhand des Eintrags in *mProfil* die ProfilHoehe und speichert diese im Attribut *mProfilHoehe* ab. In *anzProfilReihe* wird die Anzahl der Buchstaben übergeben, welche in *mProfil* als Profilreihenbezeichnung abgelegt ist.

private void FuelleListeGuete()
Die Methode liest die Daten aus der Datei *Stahlguete.txt* in die Liste *mListeGuete* ein.

private void FuelleListeProfilDruck(String profilReihe)
Die Methode liest die Daten aus der Datei Profil_<profilReihe>_DRUCK.txt in die Liste *mListeProfil* ein.

private String FormatiereFormat1(String zeichenKette)
Die Methode entfernt die Leerzeichen am Anfang und am Ende des Strings *zeichenKette*, zusätzlich werden alle Buchstaben in Großbuchstaben umgewandelt.

private int GetIndexZiffer(String zeichenKette)
Die Methode sucht das erste Vorkommen einer Ziffer, '1' .bis.'9' im String *zeichenKette*. Der Index der Stelle wird zurückgegeben. Die Zählung beginnt mit 0!

private boolean IntegerKompatibel(String testString)
Die Methode überprüft, ob die durch *testString* übergebene Zeichenkette in einen gültigen Integer Wert umgewandelt werden kann. Kann der String in ein Integer Wert umgewandelt werden, wird *true* zurückgeliefert, ansonsten *false*

private void SetzeProfilWerte(String profilNennHoehe)
Die Methode liest zur übergebenen *profilNennHoehe* alle in der Datei enthaltenen Werte und schreibt diese in die Variablen *mQuerschnitt*, *profilQuerschnitt*, *mHoehe*, *mBreite*, *mTg*, *mIz*, und *mIy*.

private void BerechneSchlankheitsGrade()
Die Methode berechnet die Werte von *mBezogenerSchlankheitsGrad, mSchlankheitsGradY*, und *mSchlankheitsGradZ* und setzt die Variablen.

private void BerechneAbminderungsFaktor()
Die Methode berechnet den Wert von *mAbminderungsFaktor* und setzt die Variable.

### 2.6.2.2 Die Bibliothek *edtra_BDKJ.dll*

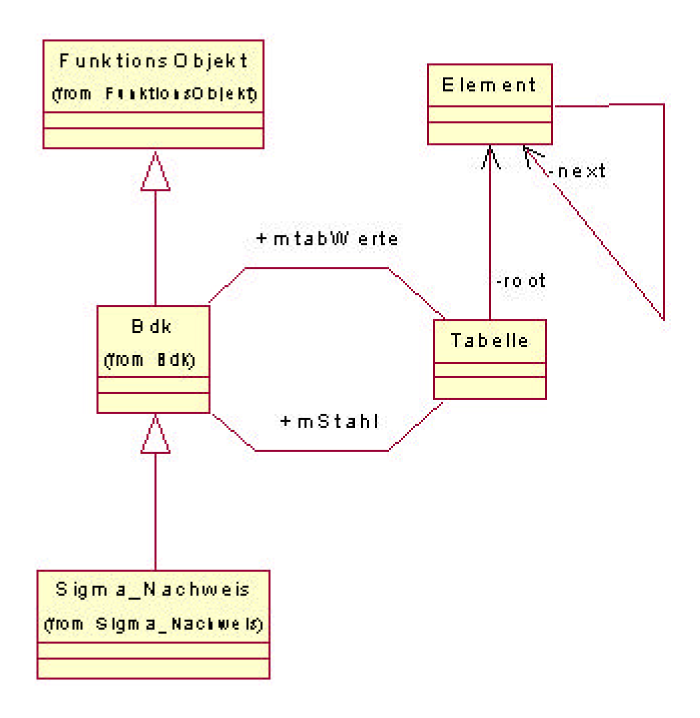



Bild 88. UML Diagramm der Bibliothek *edtra_BDKJ.dll*

Die Klassen lassen sich unterscheiden in Klassen, welche exportiert werden und anschließend als COM Objekt verwendet werden können, und Klassen, welche nicht exportiert werden und nur innerhalb der Klassenhierarchie verwendet werden.

Folgende Klassen werden nur intern verwendet:

- FunktionsObjekt.java
- Tabelle.java
- Element.java

Zu den exportierten Klassen zählen:

- Bdk.java ( COM Objekt → Bdk )
- SigmaNachweis.java ( COM Objekt → SigmaNachweis )

**Die Klasse *FunktionsObjekt***

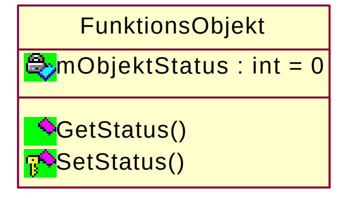



Bild 89. UML Darstellung der Klasse *FunktionsObjekt*

### Attribute

int mObjektStatus

*mObjektStatus* ist eine *private* Variable die den Status des Objekts angibt, d.h. beispielsweise, ob eine Berechnung erfolgreich durchgeführt wurde oder nicht. Da die Variable in einer übergeordneten Klasse deklariert wurde steht *mObjektStatus* in allen abgeleiteten Klassen zur Verfügung

### Methoden

public int GetStatus ()

Die Methode dient zum Auslesen des Status des Objekts

protected int SetStatus (int status)

Die Methode dient zum Verändern des Status des Objekts

### Die Klasse *Tabelle*

Die Klasse Tabelle verwaltet die Daten aus einer Datei. Die Daten werden als eine verkettete Liste abgespeichert.

```
Tabelle
-----------------------------
status : int
OK : int = 1
ERROR : int = 0
EMPTY : int = -1
FILE_NOT_EXIST : int = -2
aktuell : int
anzahl : int
root : Element
-----------------------------
Tabelle()
getWert()
create()
getStatus()
nextExist()
getName()
```

**Attribute**

```
private int status              // Zeigt den Status der Klasse an
private int OK = 1              // Konstanten Wert für den Status
private int ERROR = 0           // Konstante für Fehleranzeige
private int EMPTY = -1          // Konst. für fehlende Werte
private int FILE_NOT_EXIST = -2 // Konst. für fehlende Textdatei
private int aktuell             // Interne Zähler für die Ausgabe der
                                // Überschrift. Nummer des aktuellen
                                // Elements in der Liste
private int anzahl              // Anzahl der Elemente
private Element root// Das erste Element in der Verketteten Liste
```

Bild 90. UML Darstellung der Klasse *Tabelle*

### Methoden

public Tabelle(String path)
Konstruktor der Klasse *Tabelle*. Parameter *path* beinhaltet den absoluten Pfad zu der Datei, die eingelesen werden soll. Im Konstruktor werden die einzelnen Zeilen der Textdateien in einer verketteten Liste in je ein Objekt vom Typ *Element* eingelesen.

public double getWert(String zeile, int spalte)

Mit *Zeile* wird ein Schlüssel übergeben, mit dem auf die korrekte Zeile in der Tabelle zugegriffen werden soll. *Spalte* beinhaltet den Spaltenindex des Werts, der ausgelesen werden soll. Die Methode gibt ein Wert aus der Tabelle zurück.

private Element create(String zeile)
Gibt eine leere Instanz der Klasse *Element* zurück, bzw. generiert eine neue Instanz der Klasse.

public int getStatus()
Gibt Status der Klasse zurück.

public int nextExist()
Die Methode gibt *OK* zurück falls eine weitere Zeile existiert.

public String getName()
Die Methode gibt den Namen der aktuellen Spalte zurück.

## Die Klasse *Element*

Die Klasse Element verwaltet eine Zeile aus der Wertetabelle.

### Attribute

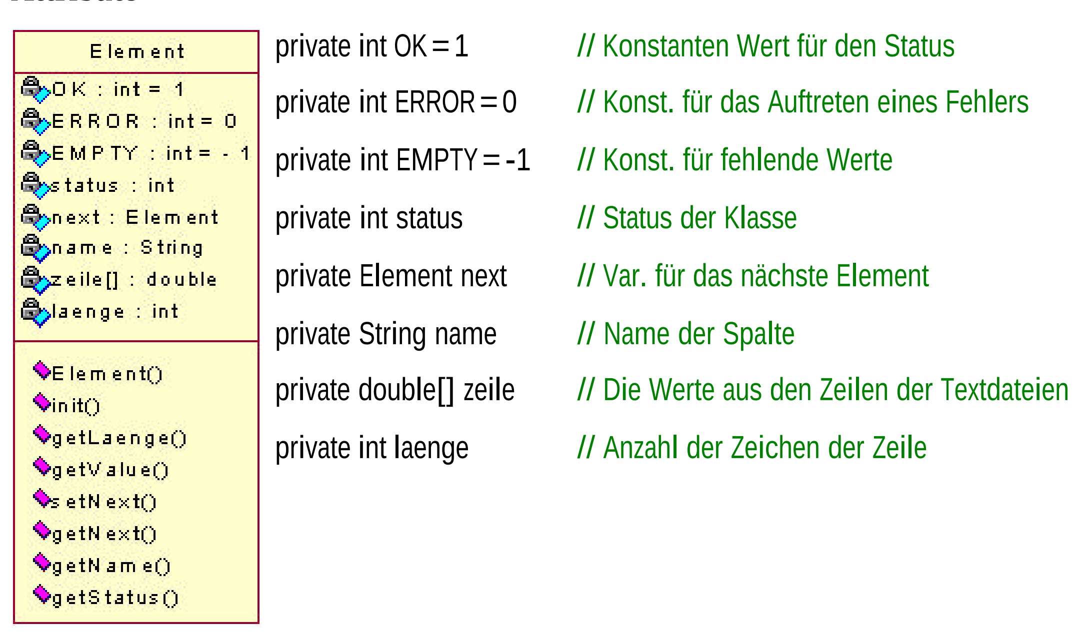



Bild 91. UML Darstellung der Klasse *Element*

**Methoden**

public Element()
Default Konstruktor. Hier werden bei der Objekterzeugung verschiedene Variablen, wie beispielsweise *status* mit Werten belegt

public int init(String zeileStr)
In der Methode *init* werden die unterschiedlichen Werte einer Zeile, die mit *zeileStr* als String übergeben wird, in ein Array vom Typ Double in einer bestimmten Reihenfolge eingelesen. Der Parameter *zeileStr* beinhaltet die Überschrift und die Werte der Zeile, die gelesen werden sollen. Mit Überschrift der Zeile ist die erste Zeichenfolge gemeint, wie beispielsweise IPE300.

public int getLaenge(){return laenge;}
Die Methode gibt die Länge des Array mit Werten zurück.

public double getValue(int spalte)
Die Methode gibt den Wert aus einer bestimmten Spalte des Array mit den Ergebniswerten zurück. Der Zähler beginn bei 1.

public int setNext(Element nextElem)
Die Methode speichert das nächste Element.

public Element getNext()
Die Methode gibt das nächste Element zurück

public String getName()
Die Methode gibt die Überschrift der Zeile zurück.

public int getStatus()
Die Methode gibt den Status der Klasse zurück.

public int reset()
Die Methode setz die Werte, die in der Klasse gespeichert waren zurück.

**Die Klasse *Bdk***

Da die Klasse Bdk alleine 85 Attribute enthält, werden diese aus Platzgründen im UML Diagramm der Klasse, siehe Bild 92, nicht explizit angezeigt. Eine Auflistung der Attribute erfolgt jedoch anschließend. Die Klasse beinhaltet die Funktionalität für Stabilitätsnachweise des Biegeknickens, sowie des Biegedrillknickens nach Theorie II. Ordnung, für einachsige Biegung mit und ohne Normalkraft und zweiachsige Biegung mit und ohne Normalkraft. Für zweiachsige Biegung wurden die Stabilitätsnachweise sowohl nach Methode I, sowie auch nach Methode II implementiert.

Für das Biegeknicken und Biegedrillknicken wurden insgesamt 4 unterschiedliche Methoden programmiert. Hierbei wurden unterschiedliche Gleichungen aus Kapitel 2.6.1.3 verwendet, die im folgenden aufgelistet sind.

### Biegeknicken und Biegedrillknicken für einachsige Biegung ohne Normalkraft

Funktion: biegungEinachsigOhneN
Für den Nachweis Verwendung fanden die Gleichungen Gl. (122)bis Gl. (129), wobei verschiedene Hilfswerte benötigt wurden:
- Bezogener Schlankheitsgrad $\overline{l_K}$ nach Gl. (123)
- Abminderungsfaktor $k$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127).
- Momentenbeiwert $b_m$ für Biegeknicken nach Tabelle 10

### Biegeknicken und Biegedrillknicken für einachsige Biegung mit Normalkraft

Funktion: biegungEinachsig
Für den Nachweis Verwendung fanden die Gleichungen Gl. (130)bis Gl. (138), wobei verschiedene Hilfswerte benötigt wurden:
- Bezogener Schlankheitsgrad $\overline{l_K}$ nach Gl. (123)
- Abminderungsfaktor $k$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127).
- Abminderungsfaktor $k_M$ für das Biegedrillknicken nach Gl. (127) bis Gl. (128)
- $b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ nach Tabelle 10, Spalte 3

### Biegeknicken und Biegedrillknicken für zweiachsige Biegung mit und ohne Normalkraft nach Nachweismethode 1

Funktion: biegungZweiachsig
Für den Nachweis Verwendung fanden die Gleichungen Gl. (139) bis Gl. (143), wobei verschiedene Hilfswerte benötigt wurden:
Bezogener Schlankheitsgrad $\overline{l_K}$ für das Ausweichen zur jeweiligen Achse nach Gl. (123)
$\overline{l_{K,y}}$ nach Gl. (132) bis Gl. (135)
Abminderungsfaktor $k_y$ bzw. $k_z$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127)

$b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ nach Tabelle 10, Spalte 3

### Biegeknicken und Biegedrillknicken für zweiachsige Biegung mit und ohne Normalkraft nach Nachweismethode 2

Funktion: MethodeII
Für den Nachweis Verwendung fanden die Gleichungen Gl. (144) bis Gl. (147), wobei verschiedene Hilfswerte benötigt wurden:
$k_z$ Abminderungsfaktor für das Biegeknicken nach den Europäischen Knickspannungslinien in Abhängigkeit vom bezogenen Schlankheitsgrad $\overline{l}_{Kz}$ nach Gl. (125) bis Gl. (127).

$k_M$ Abminderungsfaktor für das Biegedrillknicken nach Gl. (127) bis Gl. (128).

$k_y$ Beiwert zur Berücksichtigung des Verlaufs von $M_y$ nach Gl. (136) bis Gl. (137)
$b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$

nach Tabelle 10, Spalte 3

Sämtliche Formeln stammen aus [**Fehler! Textmarke nicht definiert.**] aus dem Kapitel Stahlbau.

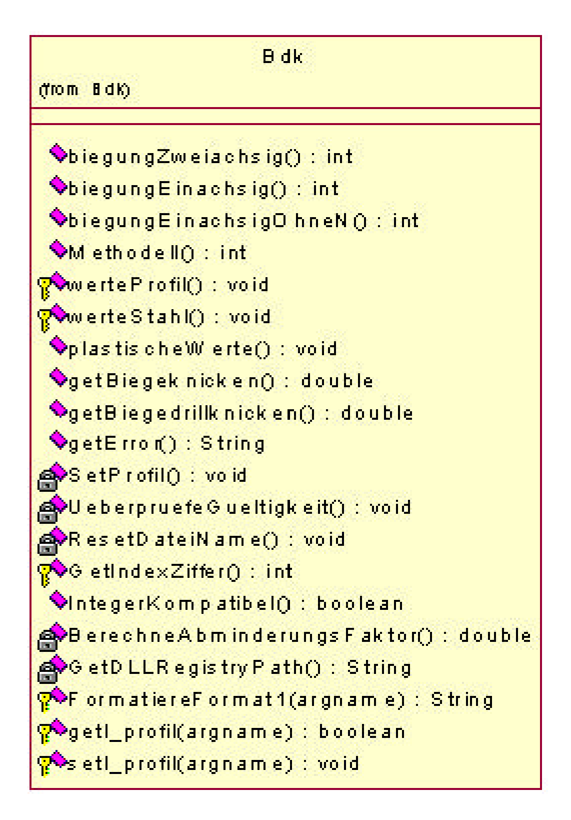



Bild 92. UML Darstellung der Klasse Bdk

**Attribute**

protected static double pi = 3.14159265359
// Statisch Konstanze Variable für die Zahl Pi

private boolean bool_Iprofil
// true, wenn I-Profil gewählt wurde, wegen Ausrundungen //stimmen die Formeln für die plastischen Werte nicht mehr

private double mAgurt
// Var. für die eine Gurtfläche eines I Profils

private double mAsteg
// Var. für die Stegfläche eines I Profils

private double zSteg
// Schwerpt.-Abstand des halben Stegs

private double zGurt
// Schwerpt.-Abstand des Gurts

protected double deltaA; // Var. zur Berücksichtigung der Ausrundung bei I Profilen

protected double deltaSy; // Var. zur Berücksichtigung der Ausrundung bei I Profilen

protected double mSy; // Flächenmoment I. Grades z-Richtung

protected double mSz; // Flächenmoment I. Grades y-Richtung

// mögliche Statuszustände beim Auslesen von Werten, siehe Klassse Tabelle

private int OK = 1 // Konstanten Wert für den Status

private int ERROR = 0 // Konst. für das Auftreten eines Fehlers

private int EMPTY = -1 // Konst. für fehlende Werte

private int FILE_NOT_EXIST = -2 // Konst. für fehlende Textdatei

private int NACHWEIS_NICHT_ERFUELLT = -4 // Für Meldung, dass Nachweis nicht erfüllt ist

// Variablen für die Belastung

private double mKappaY; // Abminderungsfaktor Kappa für Biegeknicken senkr. Y-Achse

```
private double mKappaZ;          // Abminderungsfaktor Kappa für Biegeknicken senkr. Z-Achse

private double mKappa;           // der kleinere der beiden Werte (KappaY | KappaZ) wird für
                                 //den Nachweis maßgebend

private double mKappaM                 // Abminderungsfaktor Biegemomente beim
Biegedrillknicknachweis

private double mBezogenerSchlankheitsGrad      // Variable für den bezogenen Schlankheitsgrad λK_quer

private double mSchlankheitsGradY              // Variable für den Schlankheitsgrad des Stabes für das
                                       // Biegeknicken um die Y-Achse

private double mSchlankheitsGradZ              // Variable für den Schlankheitsgrad des Stabes für das
                                       // Biegeknicken um die Z-Achse

protected double mWply                 // plast. Widerstandsmoment Y-Achse

protected double mWplz                 // plast. Widerstandsmoment Z-Achse

private double mAlphaPl_Y              // Plastischer Formbeiwert Wpl=Wel*alpha_pl für Biegung um
Y

private double mAlphaPl_Z              // Plastischer Formbeiwert Wpl=Wel*alpha_pl für Biegung um
Z

protected double mNd                   // vorhandene Normalkraft

protected double mMyd                  // vorhandenes Moment um Y

protected double mMzd                  // vorhandenes Moment um Z

protected double mNpl                  // max aufnehmbare plastische Normalkraft

protected double mMply                 // max plast. Moment um die Y-Achse

protected double mMplz                 // max plast. Moment um die Z-Achse

private double mNkiyd                  // zur Berechnung des idealen Biegedrillknickmoments

private double mNkizd                  // zur Berechnung des idealen Biegedrillknickmoments

private double mMkzd                   // zur Berechnung des idealen Biegedrillknickmoments

private double mMkyd                   // Ideales Biegedrillknickmoment

private double mEmodul=210000000       // E-Modul [kN/m ^ 2]

private double mBiegeknicken           // Ergebnis für Biegeknicksicherheit

private double mBiegedrillknicken              // Ergebnis für Biegedrillknicksicherheit

private double mC;                     // Beiwert für ideales Biegedrillknickmoment
                                       // bzw. Drehradius des Querschnitts

private double mMkiyd;                 // Ideales Biegedrillknickmoment

// Für mehrere Methoden

private String mProfil                 // Profilbezeichnung
```

```
private double mProfilHoehe         // ProfilNennHoehe
private double mQuerschnitt         // ProfilQuerschnitt
protected double mHoehe             // Hoehe
protected Integer hoehe_            // Höhe des Profils
protected double mBreite            // Breite des Profils
protected double mTg                // Flanschstaerke des Profils
private double mTraegheitsradY      // Trägheitsradius y
private double mTraegheitsradZ      // Trägheitsradius z
private double mIz                  // Trägheitsmoment II. Ord. Z
private double mIy                  // Trägheitsmoment II. Ord. Y
private double mIw                  // Wölbflächenmoment II. Grades
private double mIt                  // Torsionsflächenmoment II. Grades
private double mWy                  // Widerstandsmoment y
protected double mWz                // Widerstandsmoment z
protected double mSteg              // Stegbreite des Trägers
private double mR                   // Ausrundungsradius des Profils
private double mBetaZ               // Beta Wert für Knicklänge skz
private double mBetaY               // Beta Wert für Knicklänge sky
private double mStabLaenge          // Trägerlänge
private double mBetaMy_knick        // Momentenbeiwert für Biegeknicken nach Tafel 8.48[1]
private double mBetaMz_knick        // Momentenbeiwert für Biegeknicken nach Tafel 8.48[1]
private double mBetaMy_drill        // Momentenbeiwert für Biegeknicken nach Tafel 8.48[1]
private double mBetaMz_drill        // Momentenbeiwert für Biegeknicken nach Tafel 8.48[1]
protected double mGammaM = 1.1      // Sicherheitsbeiwert für Material
private double mZp;                 // Abstand Lastangriffspunkt der Querbelastung für Mkid
private double mXi;                 // Momentenbeiwert für ideale Biegedrillknickmoment Mkid
private double mSky                 // Knickläng für Ausweichen senkr. zur Y-Achse
private double mSkz                 // Knickläng für Ausweichen senkr. zur Z-Achse
private double mLambdaKy_quer// bezogener Schlankheitsgrad Y Biegeknicken 2-Achsig mit Normalkraft
private double mLambdaKz_quer// bezogener Schlankheitsgrad Z Biegeknicken 2-Achsig mit Normalkraft
private double mLambdaK_quer // maßgebender der beiden Werte mLambdaKz_quer; mLambdaKy_quer
```

[1] Die Angaben zu Tafeln sind bezogen auf die Schneider Bautabellen, 13. Auflage, Werner Verlag, Düsseldorf

private double mLambdaM_quer // Bezogener Schlankheitsgrad für Biegedrillknicken

private double mLambdaA // Bezugsschlankheitsgrad (Abhängig von Stahlsorte)

private double mTraegerbeiwert_n // Traegerbeiwert für KappaM

private double mAlpha // Beiwert für kaY zur Berücksichtigung des Momentenverlaufs

private double mAlphaZ // Beiwert für kaZ zur Berücksichtigung des Momentenverlaufs

private double mKaY // Beiwert k zur Berücksichtigung des Momentenverlaufs

private double mKaZ // Beiwert k zur Berücksichtigung des Momentenverlaufs

private double mDeltaN // Für Nachweis einachsige Biegung mit Druck

private String mDateiName // Zeichenfolge für die Datei aus der gelesen werden soll

private double mStreckGrenze // Variable für die Streckgrenze des Profils

protected double mStahlGuete // Var. für die Stahlgüte des Profils

private Tabelle mtabWerte // Objektvariable vom Typ Tabelle. Enthält die Profilwerte

private Tabelle mStahl // Objektvariable vom Typ Tabelle. Enthält die Werte der Streckgrenze

## Methoden

public int biegungZweiachsig(double beansprN, double beansprMy, double beansprMz,String profil, String stahlGuete, double stabLaenge, double betaY, double betaZ, double betaMy_knick, double betaMz_knick, double betaMy_drill, double betaMz_drill, double zp, double Xi)

Die Methode führt den Nachweis auf Biegeknicken und Biegedrillknicken, zweiachsige Biegung mit und ohne Normalkraft anhand der übergebenen Werte *BeansprN* (double, in kN), *BeansprMy* (double in kNm), *beansprMz* (double in kNm), *profil* (string), *stahlGuete* (String), *stabLaenge* (double in m), *betaY* (dimensionsloser double Wert), *betaZ* (dimensionsloser double Wert), *betaMy_knick* (dimensionsloser double Wert ), *double betaMz_knick* (dimensionsloser double Wert), *betaMy_drill* (dimensionsloser double Wert), *betaMz_drill* (dimensionsloser double Wert), *zp*, dem Abstand Lastangriffspunkt Querbelastung vom Schwerpunkt für *Mkid* in Meter, *Xi*, dem dimensionslosen Beiwert für den Momentenverlauf nach Tafel 8.43aus [10].

Für den Nachweis Verwendung fanden die Gleichungen Gl. (139) bis Gl. (143), wobei verschiedene Hilfswerte benötigt wurden:

Bezogener Schlankheitsgrad $\overline{l}_K$ für das Ausweichen zur jeweiligen Achse nach Gl. (123)

$\overline{l}_{K,y}$ nach Gl. (132) bis Gl. (135)

Abminderungsfaktor $k_y$ bzw. $k_z$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127)

$b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ nach Tabelle 10, Spalte 3

Sämtliche Formeln stammen aus [10] aus dem Kapitel Stahlbau.

Bemerkung:
Bei *stahlGuete* und *profil* werden die Groß- und Kleinschreibung sowie führende und abschließende Leerzeichen nicht beachtet. Die Grundlagen für die Ermittlung der Knickspannungslinie und somit dem Abminderungsfaktor sind auf Seite 8.36 und für das Biegeknicken und Biegedrillknicken nach Abschnitt 4.1.5 auf den Seiten 8.50 und 8.51 in [10] dargestellt.. Als Rückgabewert wird ein Integer Wert geliefert, der anzeigt, ob die Berechnung erfolgreich war.

```
public int biegungEinachsig(double beansprN, double beansprMy, String profil, String stahlGuete,
                            double stabLaenge, double betaY, double betaZ, double betaMy_knick,
                            double betaMy_drill, double zp, double Xi)
```

Die Methode führt den Nachweis auf Biegeknicken und Biegedrillknicken bei einachsiger Biegung mit Normalkraft anhand der übergebenen Werte *BeansprN* (double, in kN), *BeansprMy* (double in kNm), *profil* (string), *stahlGuete* (String), *stabLaenge* (double in m), *betaY* (dimensionsloser double Wert), *betaZ* (dimensionsloser double Wert), *betaMy_knick* (dimensionsloser double Wert), *betaMy_drill* (dimensionsloser double Wert), *zp*, dem Abstand Lastangriffspunkt Querbelastung vom Schwerpunkt für *Mkid* in Meter, *Xi*, dem dimensionslosen Beiwert für den Momentenverlauf nach Tafel 8.43a in[10]. Es gelten dieselben Bemerkungen wie für die Methode *biegungZweiachsig.*

Für den Nachweis Verwendung fanden die Gleichungen Gl. (130)bis Gl. (138), wobei verschiedene Hilfswerte benötigt wurden:

- Bezogener Schlankheitsgrad $\overline{l_K}$ nach Gl. (123)
- Abminderungsfaktor $k$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127).
- Abminderungsfaktor $k_M$ für das Biegedrillknicken nach Gl. (127) bis Gl. (128)
- $b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ nach Tabelle 10, Spalte 3

```
public int biegungEinachsigOhneN(double beansprMy, String profil, String stahlGuete, double stabLaenge,
                  double betaY, double betaZ, double betaY_knick, double betaY_drill, double zp, double
Xi)
```

Die Methode führt den Nachweis auf Biegeknicken und Biegedrillknicken bei einachsiger Biegung ohne Normalkraft, anhand der uebergebenen Werte *beansprMy* (double in kNm), *profil* (String), *stahlGuete* (String), *stabLaenge* (double in m), *betaY* (dimensionsloser double Wert), *betaZ* (dimensionsloser double Wert), *zp*, dem Abstand Lastangriffspunkt der Querbelastung vom Schwerpunkt für *Mkid* in Meter, *Xi*, dem dimensionslosen Beiwert für den Momentenverlauf nach Tafel 8.43a in [10]. Es gelten dieselben Bemerkungen wie für die Methode *biegungZweiachsig.*

Für den Nachweis Verwendung fanden die Gleichungen Gl. (122)bis Gl. (129), wobei verschiedene Hilfswerte benötigt wurden:

- Bezogener Schlankheitsgrad $\overline{l_K}$ nach Gl. (123)
- Abminderungsfaktor $k$ nach den europäischen Knickspannungslinien nach Gl. (125) bis Gl. (127).
- Momentenbeiwert $b_m$ für Biegeknicken nach Tabelle 10

Sämtliche Formeln stammen aus [10] aus dem Kapitel Stahlbau.

public int MethodeII(double beansprN, double beansprMy, double beansprMz, String profil, String stahlGuete,
double stabLaenge, double betaY, double betaZ, double betaMy_knick, double betaMz_knick,
double betaMy_drill, double betaMz_drill, double zp, double Xi)

Die Methode führt den Nachweis auf Biegeknicken und Biegedrillknicken, zweiachsige Biegung mit und ohne Normalkraft nach Nachweismethode II. Die Übergabeparameter und Bemerkungen entsprechen der Methode *biegungZweiachsig* und werden an dieser Stelle nicht mehr aufgeführt.

Für den Nachweis Verwendung fanden die Gleichungen Gl. (144) bis Gl. (147), wobei verschiedene Hilfswerte benötigt wurden:

$k_z$ Abminderungsfaktor für das Biegeknicken nach den Europäischen Knickspannungslinien in Abhängigkeit vom bezogenen Schlankheitsgrad $\bar{l}_{Kz}$ nach Gl. (125) bis Gl. (127).

$k_M$ Abminderungsfaktor für das Biegedrillknicken nach Gl. (127) bis Gl. (128).

$k_y$ Beiwert zur Berücksichtigung des Verlaufs von $M_y$ nach Gl. (136) bis Gl. (137)

$b_{m.y}$ Momentenbeiwert für das Biegeknicken zur Erfassung des Momentenverlaufs $M_{y,d}$ nach Tabelle 10, Spalte 3

Sämtliche Formeln stammen aus [10 ] aus dem Kapitel Stahlbau.

protected void werteProfil(String profilReihe,String profilNennHoehe)

Die Methode liest die Tafelwerte der Profile in die Variablen der Klasse ein.

protected void werteStahl(String stahlGuete)

Die Methode liest den Werte der Streckgrenze des Stahls in die Variable *mStahlGuete* ein.

public void plastischeWerte(boolean profil)

Die Methode berechnet die plastischen Querschnittswerte eines Profils. Berechnung der Flächenmomente I. Grads für *Wply*, *Wplz* und *Sy*.

public double getBiegeknicken()

Die Methode dient zum Auslesen der Werte für den Biegeknicknachweis.

public double getBiegedrillknicken()

Die Methode dient zum Auslesen der Werte für den Biegedrillknicknachweis.

public String getError(int errNumber)

Die Methode liefert für die übergebene Fehlernummer *errNumber* den dazugehörenden Fehlertext als String zurück. Die Werte und die zugehörige Fehlermeldung entsprechendem Status von *mObjectStatus*.

private void SetProfil(String profilReihe, int profilNennHoehe)

Die Methode bildet einen String, welcher der *profilReihe* beginnt und an den die *profilNennHoehe* angehängt wird. Dieser String wird dann in die Variable *mProfil* geschrieben.

private void UeberpruefeGueltigkeit(double beanspruchung)

Die Methode überprüft ob der Übergebene Wert gültig oder brauchbar ist. Also, ob beispielsweise die übergebene Beanspruchung zulässig ist. Ist der übergebene Wert ungültig wird *mObjectStatus* auf -1 gesetzt, sonst auf 0.

private void ResetDateiName()
Die Methode setzt den Pfad in *mDateiName* auf "" und löscht somit vorherige Angaben.

protected int GetIndexZiffer(String zeichenKette)
Die Methode sucht das erste Vorkommen einer Ziffer, '1' .bis.'9' im String *zeichenKette.* Der Index der Stelle wird zurückgegeben. Die Zaehlung beginnt mit 0!

public boolean IntegerKompatibel(String testString)
Die Methode überprüft, ob die durch *testString* übergebene Zeichenkette in einen gültigen Integer Wert umgewandelt werden kann. Kann der String in ein Integer Wert umgewandelt werden, wird *true* zurückgeliefert, ansonsten *false*

private double BerechneAbminderungsFaktor(double schlankheitsGrad,char Richtung)
Die Methode berechnet den Wert von *mAbminderungsFaktor* und *Richtung* den Abminderungsfaktor Kappa und setzt die Variable *kappa*. Dieser wird als Rückgabewert geliefert.

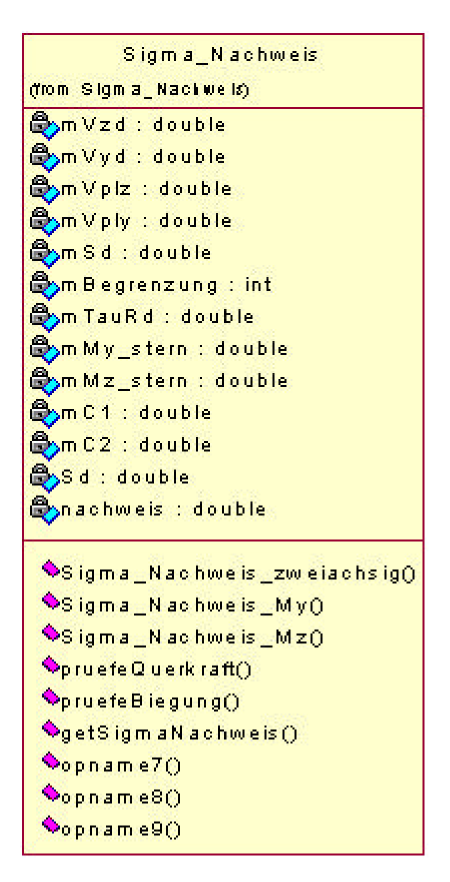
protected String GetDLLRegistryPath()
Diese Methode liest den Pfad aus der Windows-Registry unter dem die DLL installiert wurde. Das macht das Lesen der Tabellendateien unabhängig vom Ort der Installation. Der aktuelle Schlüssel lautet // 36C05CE4-0806-11D5-92FB-00A0CCD4D948.

protected String FormatiereFormat1(String zeichenKette)
Die Methode entfernt die Leerzeichen am Anfang und am Ende des Strings *zeichenKette*, zusätzlich werden alle Buchstaben in Großbuchstaben umgewandelt.

protected boolean getI_profil()
Die Methode dient zum Auslesen, ob es sich bei dem Übergebenen Profil um ein I-Profil handelt. Hier gelten die verwendeten Formeln nicht mehr. Die Methode liefert wahr zurück, wenn es sich um ein I-Profil handelt.

protected void setI_profil()
Die Methode dient zum setzen des Bool'schen Werts *bool_Iprofil*. Wegen der Ausrundungen stimmen die Formeln für die Ermittlung der plastischen Werte nicht mehr. Die Variable ist auf *true* gesetzt, wenn es sich um ein I-Profil handelt.

## Die Klasse *Sigma_Nachweis*




Die Klasse beinhaltet die Funktionalität für die gewöhnlichen Spannungsnachweise nach dem Verfahren **elastisch-plastisch** für nicht stabilitätsgefährdeten Bauteilen.

Bild 93. UML Diagramm der Klasse Sigma_Nachweis

Für die Spannungsnachweise wurden insgesamt 3 unterschiedliche Methoden programmiert. Hierbei wurden unterschiedliche Gleichungen aus Kapitel 2.6.1.2.2 verwendet, die im folgenden aufgelistet sind.

**Vereinfachter Tragsicherheitsnachweis für doppelt-symmetrische I-Profile mit den Beanspruchungen $N, M_y, V_z$**

Funktion: Sigma_Nachweis_My

Für den Nachweis Verwendung fand Tabelle 6, welcher die Gleichungen Gl. (110) bis Gl. (112) Verwendung fanden.

**Vereinfachter Tragsicherheitsnachweis für doppelt-symmetrische I-Profile mit den Beanspruchungen *N, Mz, Vy***

Funktion: Sigma_Nachweis_Mz

Für den Nachweis Verwendung fand Tabelle 7, welcher die Gleichungen Gl. (110), sowie Gl. (113) und Gl. (114) Verwendung fanden.

**Tragsicherheitsnachweis für doppeltsymmetrische I-Profile mit den Beanspruchungen $N, M_y, V_z, M_z, V_y$**

Funktion: Sigma_Nachweis_zweiachsig

Für den Nachweis Verwendung fanden die Gleichungen Gl. (115)bis Gl. (120).

### Attribute

```
private double mVzd       // vorh. Querkraftkraft in z-Richtung
private double mVyd       // vorh. Querkraftkraft in y-Richtung
private double mVplz      // plast. Querkraft des Trägers z-Richtung
private double mVply      // plast. Querkraft des Trägers y-Richtung
private double mSd        // Nachweis,auslesen mit getSigma_Nachweis
private int mBegrenzung   // Wenn=1 muss Mplz auf 1.25
                          //*SigmaRd*Wel begrenzt werden
private double mTauRd;    // Grenzschubspannung
private double mMy_stern        // Diese vier Werte dienen dem
private double mMz_stern        // Nachweis bei Biegung um
private double mC1              // zwei Achsen mit Querkraft-
private double mC2              // und Normalkraftbeanspruchung
private double Sd               // Ermitteltes Ergebnis des Nachweises
private double nachweis         // Var. zum Auslesen des Ergebnisses
```

### Methoden

```
public int Sigma_Nachweis_zweiachsig(double Nd, double Myd, double Mzd, double Vzd,
                    double Vyd, String profil, String stahl,int begrenzung)
```

Die Methode führt den Spannungsnachweis bei Belastung des I-förmigen Querschnitts mit Momenten um zwei Achsen, Querkraftbeanspruchung in y- sowie z-Richtung und Normalkräften in Stablängsrichtung. Die Formeln wurden nach [**Fehler! Textmarke nicht definiert.**] programmiert. Die Übergabeparameter sind die Normalkraft $N_d$ in [kN] als double, das Moment um die y-Achse $M_{yd}$ in [kN/m] als double, das Moment um die z-Achse $M_{zd}$ in [kN/m] als double, die Querkraft $V_{zd}$ in Z-Richtung in [kN] als double, die Querkraft $V_{yd}$ in y-Richtung in [kN] als double, die Profilbezeichnung *profil* als String, die Bezeichnung der Stahlgüte *stahl* als String und der dimensionslose Integerwert *begrenzung* für die Begrenzung von *Mplz* auf den 1.25-fachen Wert von $\sigma_{Rd}*W_{el}$. Rückgabewert ist ein Integer, der mit dem Wert 0 anzeigt, dass die Berechnung erfolgreich war.

```
public int Sigma_Nachweis_My(double Nd, double Myd, double Vzd, String profil, String stahl,int
begrenzung)
```

Die Methode führt den allgemeinen Spannungsnachweis für die Biegung um die y-Achse. Die Übergabeparameter sind die Normalkraft $N_d$ in [kN] als double, das Moment um die y-Achse $M_{yd}$ in [kN/m] als double, die Querkraft $V_{zd}$ in z-Richtung in [kN] als double, die Profilbezeichnung *profil* als String, die Bezeichnung der Stahlgüte *stahl* als String und der dimensionslose Integerwert *begrenzung* für die Begrenzung von *Mplz* auf den 1.25-fachen Wert von $\sigma_{Rd}*W_{el}$. Rückgabewert ist ein Integer, der mit dem Wert 0 anzeigt, dass die Berechnung erfolgreich war.

```
public int Sigma_Nachweis_Mz(double Nd, double Mzd, double Vyd, String profil, String stahl,int
begrenzung)
```

Die Methode führt den allgemeinen Spannungsnachweis für die Biegung um die z-Achse. Die Übergabeparameter sind die Normalkraft $N_d$ in [kN] als double, das Moment um die z-Achse $M_{zd}$ in [kN/m] als double, die Querkraft $V_{yd}$ in y-Richtung in [kN] als double, die Profilbezeichnung *profil* als String, die Bezeichnung der Stahlgüte *stahl* als String und der dimensionslose Integerwert *begrenzung* für die Begrenzung von *Mplz* auf den 1.25-fachen Wert von $\sigma_{Rd}*W_{el}$. Rückgabewert ist ein Integer, der mit dem Wert 0 anzeigt, dass die Berechnung erfolgreich war.

private int pruefeQuerkraft(){
Die Methode überprüft, ob der Nachweis zweiachsig mit Normalkraft geführt werden kann, d.h. ob die Interaktionsformeln für den Spannungszustand angewendet werden dürfen.

private void pruefeBiegung()
Die Methode überprüft, ob es sich um zweiachsige Biegung handelt oder nicht. Hier gelten andere Anwendungskriterien als bei einachsiger Biegung.

public double getSigmaNachweis()
Die Methode dient zum Auslesen des ermittelten Wertes für den Nachweis.

### 2.6.2.3 Die Bibliothek *edtra_PrfJ.dll*

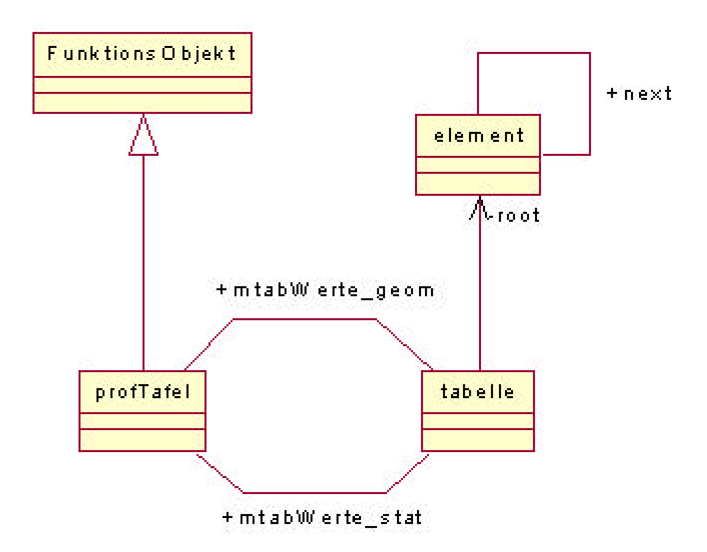



Bild 94. UML Diagramm der Bibliothek *edtra_PrfJ.dll*

Die Bibliothek *edtra_PrfJ.dll* beinhaltet die Funktionalität zum Auslesen von Werten von normierten Walzprofilen aus dem Programm der Firma Arbed [H1]. Die Firma Arbed stellt eine Diskette zur Verfügung in denen die Werte für gebräuchliche normierte Walzprofile als ASCII-Code gespeichert sind. Die Dateien sind wie folgt definiert:

*GEOM.DAT: Enthält die geometrischen Werte der Profile und entspricht den linken Seiten des PROFILARBED Verkaufskatalogs.

*STAT.DAT: Enthält die statischen Werte der Profile und entspricht den rechten Seiten des PROFILARBED Verkaufskatalogs.

Folgende Profilserien waren auf der vorliegenden Diskette aufgeführt:

| | |
|---|---|
| IPE | I-Profile gemäss EU 19-57, inklusive der abgeleiteten Profile A und O sowie der Profile IPE 750 |
| HE | Breitflanschträger gemäss EU 53-62, inklusive der abgeleiteten Profile AA und der Profile HL mit extrabreiten Flanschen |
| HD | Breitflansch-Stützenprofile |
| HP | Breitflanschpfähle |
| W | Amerikanische Breitflanschträger gemäss ASTM A6/A 6M-93b |
| UB | Britische Universalträger gemäss BS4 part 1 - 1993 |
| UC | Britische Universalstützen gemäss BS4 part 1 - 1993 |
| IPN | Europäische Normalträger (Flanschneigung : 14%) |
| UAP | U-Profile mit parallelen Flanschen gemäss NF A 45-255 |
| UPN | Europäische U-Stahl Normalprofile |
| L | Gleichschenkliger Winkelstahl gemäss EU 56-77 und DIN 1028 |
| L | Ungleichschenkliger Winkelstahl gemäss EU 57-78 und DIN 1029 |

Die Klassen lassen sich unterscheiden in Klassen, welche exportiert werden und anschließend als COM Objekt verwendet werden können, und Klassen, welche nicht exportiert werden und nur innerhalb der Klassenhierarchie verwendet werden.

Folgende Klassen werden nur intern verwendet:

- FunktionsObjekt.java
- Tabelle.java
- Element.java

Zu den exportierten Klassen zählen:

- profTafel.java ( COM Objekt → profTafel )

Die Klassen *FunktionsObjekt*, *Tabelle* und *element* entsprechen in Aufbau und Struktur exakt der Bibliothek *edtra_BDKJ.dll*. So wird an dieser Stelle auf eine nochmalige Abbildung der Klassen verzichtet.

Die Klasse *Tabelle* verwaltet die Daten aus einer der Textdateien, aus der Werte ausgelesen werden sollen. Die Daten werden als verkettete Liste abgespeichert. Innerhalb der Klasse befinden sich Methoden, die feststellen, ob eine Zeile der Textdatei Informationen enthält. Dies geschieht über einen Output Stream. Für die erste Zeile mit Informationen wird schließlich ein Objekt der Klasse *Element* erzeugt. Innerhalb dieser Klasse sind dann die Informationen gespeichert, die benötigt werden. In Element werden die Zahlenwerte der Profile in einem Array aus double Werten gespeichert, wobei die Zeile in die einzelnen Werte aufgespaltet wird. Weiterhin besitzt *Element* ein Attribut *name*, vom Typ Srtring, in welchem die Profilbezeichnung abgespeichert ist (die ersten 20 Zeichen einer Zeile der Textdatei). Zudem enthält die Klasse *Element* ein Attribut, für ein weiteres Objekt der eigenen Klasse (*Element*).

In *Tabelle* wird nun festgestellt, ob eine weitere Zeile mit Informationen vorhanden ist. Für die nächste Zeile wird schließlich ein Objekt der Klasse *Element* erzeugt und an das erste *Element* Objekt weiter gereicht. Innerhalb des ersten Element Objekts wird dann ein weiteres Objekt vom Typ *Element* erzeugt und die Informationen darin gespeichert. Dies geschiet nun für jede Zeile in der Textdatei. Letztlich liegt eine verkettete Liste vor, in der jedes *Element* Objet ein weiteres *Element* Objekt enthält.

Beim Auslesen der Werte wird im ersten *Element* Objekt der "Name" des Objekts (Attribut *name*) über die Methode *getName()* abgefragt. Stimmt dieser nicht mit dem gesuchten Profil überein, wird über die Methode *getNext()* (ebenfalls Klasse *Element*) das nächste Objekt eingelesen und der Name kann wieder überprüft werden. Dieser Vorgang geschieht so lange, bis das gesuchte Objekt mit dem richtigen Namen vorliegt. Wobei dann die gesuchten Werte einzeln, nacheinander über den Spaltenindex des Arrays für die Zahlenwerte ausgelesen werden.

Für nähere Angaben, was den Code betrifft und eine genaue Erläuterung, wie die verkettete Liste im einzelnen genau umgesetzt wurde, wird auf den Quellcode verwiesen. Nachfolgend erfolgt eine Gruppierung der einzelnen Methoden für die Klasse *profTafel*.

Methoden zur Prüfung vorhandener Strings:

getCharIndex()
getIndexZiffer()
vergleichName()
vergleichName2()

Methoden zur Formatierung von Strings:
FormatiereFormat1()
FormatiereFormat2()
summeName()

Methoden zur Verwaltung der Daten:
werteProfil()
ResetDateiName()
getError()
getValueCount()
getDLLRegistryPath()

Methoden zum Einlesen von Werten aus den Textdateien:
werteProfil()

Methoden zum Auslesen von Werten aus der Klasse profTafel:
getWerte()

**Die Klasse *profTafel***

Wie aus dem UML Diagramm der Bibliothek erkennbar ist, ist die Klasse abgeleitet von der Klasse *FunktionsObjekt*.

profTafel
status : int
OK : int = 1
ERROR : int = 0
EMPTY : int = -1
FILE_NOT_EXIST : int = -
exitValue[] : double
valueCount : int

profTafel()
FormatiereFormat1()
FormatiereFormat2()
getCharIndex()
GetIndexZiffer()
werteProfil()
ResetDateiName()
vergleichName()
vergleichName2()
GetDLLRegistryPath()
getWerte()
getValueCount()
getError()

**Attribute**

```
private int status                 // Zeigt den Status der Klasse an
private int OK = 1                 // Konstanten Wert für den Status
private int ERROR = 0              // Konstante für Fehleranzeige
private int EMPTY = -1             // Konst. für fehlende Werte
private int FILE_NOT_EXIST = -2    // Konst. für fehlende Textdatei
private double[] exitValue;// Array für die Ergebnisse
private    int valueCount;         // Anzahl der Werte die ausgelesen
                                   // werden sollen
```

Bild 95. UML Diagramm der Klasse *profTafel*

**Methoden**

Konstruktor

public int profTafel(String profil)
Die Methode ermittelt durch den Aufruf verschiedener anderer Methoden die statischen und geometrischen Werte in den zum übergebenen Profil *profil* gehörenden Dateien. Als Rückgabewert wird ein Integer Wert geliefert, der angibt, ob die Ermittlung der Werte erfolgreich war.

private String FormatiereFormat1(String zeichenKette)
Die Methode entfernt die Leerzeichen am Anfang und am Ende des Strings *zeichenKette*, zusätzlich werden alle Buchstaben in Großbuchstaben umgewandelt.

private String FormatiereFormat2(String zeichenKette)
Die Methode entfernt die Leerzeichen am Anfang und am Ende des Strings *zeichenKette*, zusätzlich werden alle Buchstaben in Großbuchstaben umgewandelt und weiterhin Stellen in der Zeichenkette gesucht, die keine Zahlen sind. Diese Zeichen werden für die weitere Verarbeitung auf ein sinnvoll Format gebracht und vor, sowie nach dem Zeichen ein Leerzeichen eingefügt.

private int[] getCharIndex(String zeichenKette)
Die Methode prüft ob außer Zahlen auch noch Buchstaben im übergebenen String *zeichenKette* enthalten sind. Als Rückgabewert wird ein eindimensionales Array aus Integer Werten geliefert, der im ersten Index anzeigt, wie viele Buchstaben in der Zeichenkette enthalten sind und in den restlichen Stellen, an welcher Stelle im String sich der Buchstabe befindet.

private int GetIndexZiffer(String zeichenKette)
Die Methode sucht das erste Vorkommen einer Ziffer, '1' bis '9' im String *zeichenKette*. Der Index der Stelle wird zurückgegeben. Die Zählung beginnt mit 0!

protected void werteProfil(String preName,String profilReihe)
Die Methode liest die ausgelesenen Tafelwerte der Profile in die zugehörigen Variablen ein.

private void ResetDateiName()
Die Methode setzt den Pfad in *mDateiName_stat* und *mDateiName_geom* auf "" und löscht somit vorherige Angaben.

private String vergleichName(String profilName){
Die Methode ermittelt über den übergebenen String *profilName* um welches Profil es sich handelt und liefert als Rückgabewert einen String, der später vor den Dateinamen angehängt wird um aus der richtigen Datei auszulesen.

private String vergleichName2(String preName,String profilName)
Die Methode ermittelt über den übergebenen String *profilName* um welches Profil es sich handelt und liefert als Rückgabewert einen String, der später nach dem Dateinamen angehängt wird, um aus der richtigen Datei auszulesen.

private String GetDLLRegistryPath()
Diese Methode liest den Pfad aus der Windows-Registry unter dem die DLL installiert wurde. Das macht das Lesen der Tabellendateien unabhängig vom Ort der Installation. Der aktuelle Schlüssel lautet // 15EF16E4-A3BB-11D5-92FD-00A0CCD4D948.

public double getWerte(int col)
Die Methode dient zum Auslesen der Ergebniswerte. Und liefert den zu *col* gehörenden Wert aus der Tabelle. Die Variable *col* entspricht der Spalte in der Datei, zu welcher der Wert gehört.

public String getError(int errNumber)
Die Methode liefert für die übergebene Fehlernummer *errNumber* den dazugehörenden Fehlertext als String zurück. Die Werte und die zugehörige Fehlermeldung entsprechendem Status von *mObjectStatus*.

private String summeName(String preName,String postName,String profilHoehe)
Die Methode fügt den Profilnamen in der richtigen Reihenfolge zusammen und liefert diesen als String als Ergebnis zurück.

---

## 2.7 Holzbau

Alle hier aufgeführten Funktionen wurden auf der Grundlage der „DIN 1052 – Holzbauwerke“ Teil 2 (4.88) entwickelt. Die Berechnungsformeln stammen zumeist aus [10]

### 2.7.1 Theoretische Grundlagen

#### 2.7.1.1 Normalkraftbeanspruchung

Für die Nachweise der Normalkraftbeanspruchung sind zwei Einzelnachweise zu unterscheiden:

- Druckkräfte
- Zugkräfte

Diese Zustände sind in der Programmierung für den Holzbau differenziert worden. Allgemein gilt für beide Funktionen zunächst der Nachweis über die Spannungen nach der Formel:

$$s = N/A \qquad \text{Gl. (148)}$$

mit N als Normalkraftbeanspruchung
A als Querschnittsfläche des Stabes

Dabei kann die Querschnittsfläche beliebige Form annahmen, solange gewährleistet ist, dass die Kräfte im Querschnittsschwerpunkt angreifen.

Es wird nun nachgewiesen, dass $s_{vorh} / s_{zul} \leq 1$ ist.
Die zulässigen Spannungen sind für Zug- und Druckkräfte nicht identisch und können der DIN 1052 Teil 1 Tabelle 5 entnommen werden.

Für den Druckstab ist außerdem zu untersuchen, ob sich der Stab nicht den Druckspannungen entzieht indem er ausknickt. Für diesen Nachweis sieht die DIN eine Abminderung der zulässigen Druckspannung in betracht. Es wird daher ein Faktor $w$ (*omega*) eingeführt, der einen Wert größer als 1.0 annimmt.

Für $\lambda < 100$ gilt: $w = 1 + 2\left(l/100\right)^2$, ansonsten $w = 3 \cdot \left(l/100\right)^2$.

Gl. (149)

Durch Umstellung der Formel können darüber hinaus ermittelt werden:

- die zulässige Normalkraft eines gegebenen Querschnittes,
- die erforderlichen Querschnittsfläche bei gegebener Normalkraft (Bemessung).

### 2.7.1.2 Biegung

Für die Biegung wird wieder in mehrere Bereiche unterteilt:

- reine Biegung
- Biegung mit Zug
- Biegung mit Druck

· **reine Biegung**

Allgemein gilt für die Biegung zunächst der Nachweis über die Spannungen nach der allgemeinen Formel:

$$s = \frac{M}{W} \qquad \text{Gl. (150)}$$

mit $M$ als Biegemoment
$W$ als Wiederstandsmoment

Der Nachweis der Biegung erfolgt dann über die Formel:

$$\frac{M_Y / W_{YN} + M_Z / W_{ZN}}{zul\, s_B} \leq 1 \qquad \text{Gl. (151)}$$

Bild 96. Biegebeanspruchung eines Rechteckquerschnitts

mit

$M_Y$ bzw. $M_Z$ - Bemessungsmoment
$W_{YN}$ bzw. $W_{ZN}$ - Netto Widerstandsmoment um die Y- bzw. Z-Achse
$zul\, s_B$ - Zulässige Spannungen nach DIN 1052.

· **Biegung mit Zug**

Allgemein gilt für die Ermittlung der Normalspannung Gl. (148) und für die Biegespannung Gl. (150).

Der Nachweis für die Biegung mit Zug erfolgt dann über die Formel:

$$\frac{N / A_N}{zul\, s_{ZII}} + \frac{M_Y / W_{YN} + M_Z / W_{ZN}}{zul\, s_B} \leq 1 \qquad \text{Gl. (152)}$$

Bild 97. Biegebeanspruchung eines Rechteckquerschnitts

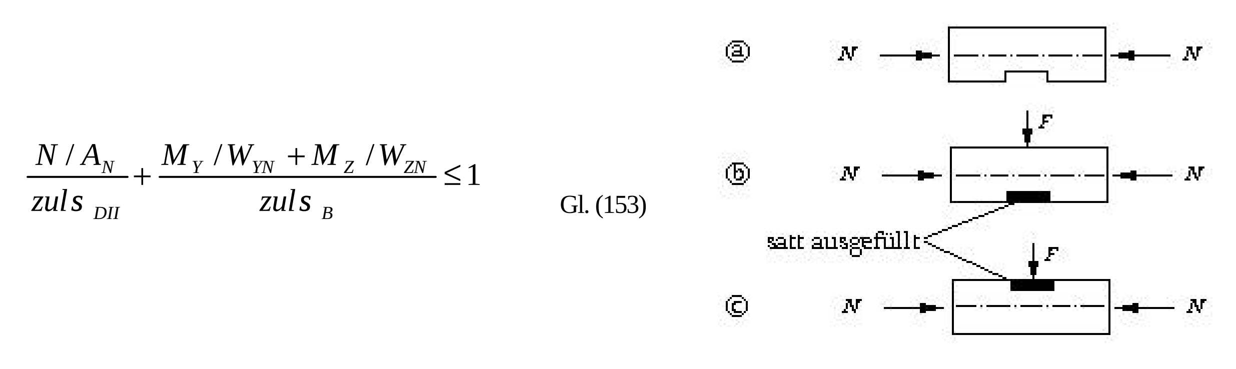
hierin bedeuten:

| | | |
|---|---|---|
| $N$ | - | Normalkraftbeanspruchung (mittige Zugkraft) |
| $A_N$ | - | Nettoquerschnitt |
| $M_Y$ bzw. $M_Z$ | - | Bemessungsmoment, z.B. exzentrischer Zug mit $M=N*e$ |
| $W_{YN}$ bzw. $W_{ZN}$ | - | Netto Widerstandsmoment um die Y- bzw. Z-Achse |
| *zuls* | - | Zulässige Spannungen nach DIN 1052. (Empirisch ermittelte Werte) |

· **Biegung mit Druck**

Allgemein gilt für die Ermittlung der Normalspannung Gl. (148) und für die Biegespannung Gl. (150).

Der Nachweis für die Biegung mit Zug erfolgt dann über die Formel:




$$\frac{N / A_N}{zul\,s_{\ DII}} + \frac{M_Y / W_{YN} + M_Z / W_{ZN}}{zul\,s_{\ B}} \leq 1 \qquad \text{Gl. (153)}$$

Bild 98. Biegebeanspruchung mit Druck

Als Querschnittswerte sind einzusetzen:
$A_N$ und $W_N$: für Fall a stets; für Fall b nur, wenn |N/A| < |M/W| ist.
$A$ und $W$: für Fall c stets; für Fall b nur, wenn |N/A| $\geq$ |M/W| ist.

hierin bedeuten:

| | | |
|---|---|---|
| $N$ | - | Normalkraftbeanspruchung (mittige Zugkraft) |
| $A_N$ | - | Nettoquerschnitt |
| $M_Y$ bzw. $M_Z$ | - | Bemessungsmoment, z.B. exzentrischer Zug mit $M=N*e$ |
| $W_{YN}$ bzw. $W_{ZN}$ | - | Netto Widerstandsmoment um die Y- bzw. Z-Achse |
| *zuls* | - | Zulässige Spannungen nach DIN 1052. (Empirisch ermittelte Werte) |

Für den Nachweis bei Druck mit Biegung ergibt sich weiterhin noch der Nachweis der Stabilität in Form eines Kippnachweises, der für die Programmierung bei der Biegung mit Druck berücksichtigt wurde. Der Nachweis erfolgt dann nach der Formel:

$$\frac{N / A_N}{zul\,s_{\ K}} + \frac{M_Y / W_{YN} + M_Z / W_{ZN}}{1.1 \times k_B \times zul\,s_{\ B}} \leq 1 \qquad \text{Gl. (154)}$$

hierin bedeuten:

N - Normalkraftbeanspruchung (mittige Zugkraft)

$A_N$ - Nettoquerschnitt

$zul\,s_K = zul\,s_{DII} / w$ mit $w$ -Werten Nach DIN 1052-1, Tab. 10 und A1

$M_Y$ bzw. $M_Z$ - Bemessungsmoment, z.B. exzentrischer Zug mit $M=N*e$

$W_{YN}$ bzw. $W_{ZN}$ - Netto Wiederstandsmoment um die Y- bzw. Z-Achse

$zul\,s_B$ - Zulässige Spannungen nach DIN 1052. (Empirisch ermittelte Werte)

$k_B$ - Kippbeiwert

$k_B$ ist abhängig vom Kippschlankheitsgrad $l_B$. Mit $l_B$ aus:

$$l_B = \sqrt{\frac{s \times h \times g_l \times zul\,s_B}{b^2 \times p \times \sqrt{E_{II} \times G_T}}} = k_B \times \sqrt{\frac{s \times h}{b^2}} \qquad \text{Gl. (155)}$$

$g_l$ = 2.0 für Lastfall H und Lastfall HZ
$E_{II}$ und $G_T$ nach DIN 1052-1 und DIN 1052-1/ A1

$k_B$ ergibt sich schließlich nach folgender Tabelle

| $l_B$ | $\leq 0.75$ | $0.75 \leq l_B \leq 1.4$ | $>1.4$ |
|---|---|---|---|
| $k_B$ | 1 | $1.56-0.75*l_B$ | $1/l_B^2$ |

Tabelle 11. Kippbeiwert kB

### 2.7.2 Programmierung in JAVA

Die Nachweise für den Holzbau sind in zwei Bibliotheken enthalten. Die Bibliothek *Edatra.dll*, enthält die Nachweise für druck- und zugbeanspruchte Stäbe ohne Biegung. Die Bibliothek *HolzDruckZugStab.dll* enthält die Nachweise für Biegung und Längskraft einschließlich der o.g. Stabilitätsnachweise.

#### 2.7.2.1 Die Bibliothek *Edatra.dll*

Die Bibliothek *edatraDLL.dll* beinhaltet die Berechnungen für die Nachweise von Holz und Stahlbauteilen für zug- und druckbeanspruchte Querschnitte. In Bild 80 sind die Klassen für die Nachweise für den Stahlbau der Übersichtlichkeit halber grau hinterlegt worden. Diese werden im Kapitel Stahlbau erläutert.

Die Klassen lassen sich unterscheiden in Klassen, welche exportiert werden und anschließend als COM Objekt verwendet werden können, und Klassen, welche nicht exportiert werden und nur innerhalb der Klassenhierarchie verwendet werden können.

Folgende Klassen werden nur intern verwendet:

- FunktionsObjekt.java
- Liste.java

Zu den exportierten Klassen zählen:

- HolzDruckStab.java ( COM Objekt → HolzDruckStab )
- HolzZugStab.java ( COM Objekt → HolzZugStab )

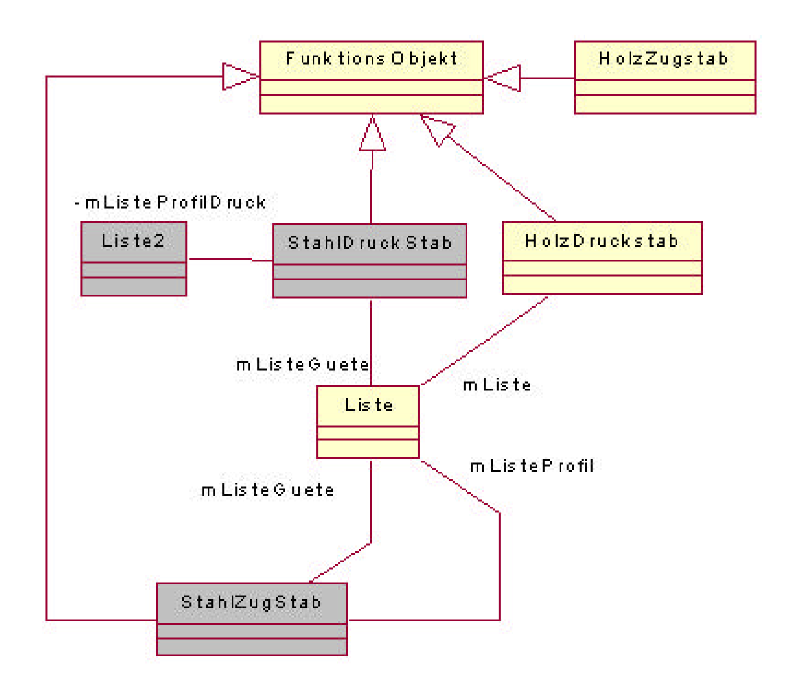



Bild 99. UML Klassendiagramm von Edatra.dll

### Die Klasse *FunktionsObjekt*

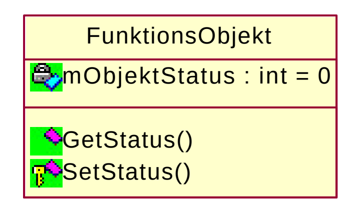



Bild 100. UML Darstellung der Klasse *FunktionsObjekt*

### Attribute

int mObjektStatus

*mObjektStatus* ist eine *private* Variable die den Status des Objekts angibt, d.h. beispielsweise, ob eine Berechnung erfolgreich durchgeführt wurde oder nicht. Da die Variable in einer übergeordneten Klasse deklariert wurde, steht *mObjektStatus* in allen abgeleiteten Klassen zur Verfügung

### Methoden

public int GetStatus ()

Die Methode dient zum Auslesen des Status des Objekts.

protected int SetStatus (int status)

Die Methode dient zum Verändern des Status des Objekts.

### Die Klasse *HolzZugstab*

| HolzZugstab |
|---|
| mConstStatusOK : int = 0<br>mConstNachweisZuGross : int = 100<br>mConstUnbekannteHolzart : int = 101<br>MConstSonstigeFehler : int = 150<br>mConstDivisionDurchNull : int = 151 |
| HolzZugstab()<br>BerechneNachweisAufZugV1()<br>BerechneNachweisAufZugV2()<br>BerechneZulaessigeBelastungV1()<br>BerechneZulaessigeBelastungV2()<br>BerechneErforderlicheBreite()<br>BerechneErforderlichenQuerschnitt()<br>BestimmeSigma() |

Bild 101. UML Darstellung der Klasse HolzZugstab

Für die Holznachweise auf Zug wurden insgesamt 8 unterschiedliche Methoden programmiert. Hierbei wurde auf Gleichungen aus Kapitel 2.7.1.1 verwendet, die im folgenden aufgelistet sind.

### Nachweise auf reinen Zug

Alle 8 Methoden basieren auf Gleichung Gl. (148).

Die Klasse ist von *FunktionsObjekt* abgeleitet, damit sind alle Attribute und Methoden der Oberklasse hier ebenfalls verfügbar.

### Attribute

Fehlerkonstanten

```
int mConstStatusOK = 0;
int mConstNachweisZuGross = 100;
int mConstUnbekannteHolzart = 101;
int mConstSonstigeFehler = 150;
int mConstDivisionDurchNull = 151;
```

### Methoden

Konstruktor:

```
public HolzZugstab()
public double BerechneNachweisAufZugV1(double beanspruchung, double breite,
                                        double hoehe, String holzSortierklasse)
```
Methode zum Berechnen des Nachweises des Zugstabes (Variante mit Breite und Höhe)

```
public double BerechneNachweisAufZugV2(double beanspruchung, double
                                        querschnitt, String holzSortierklasse)
```
Methode zum Berechnen des Nachweises des Zugstabes (Variante mit Querschnitt)

```
public double BerechneZulaessigeBelastungV1(double breite, double hoehe,
                                        String holzSortierklasse)
```
Methode zum Berechnen der zulässigen Belastung des Zugstabes
(Variante mit Breite und Höhe)

```
public double BerechneZulaessigeBelastungV2(double querschnitt,
                                        String holzSortierklasse)
```
Methode zum Berechnen der zulässigen Belastung des Zugstabes.
Variante mit Querschnitt.

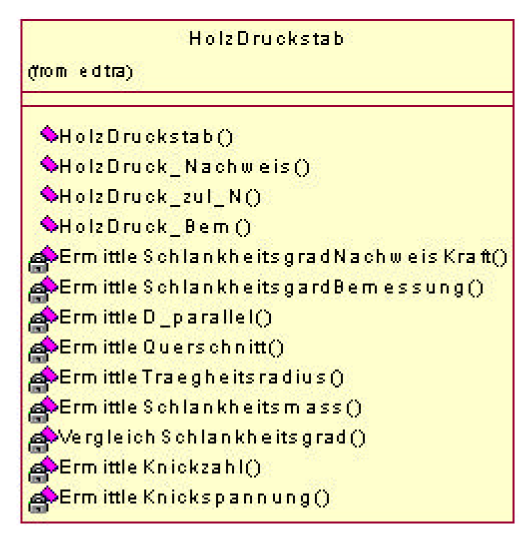
```
public double BerechneErforderlicheBreite(double beanspruchung, double
                                        hoehe, String holzSortierklasse)
```
Methode zum Berechnen der erforderlichen Breite des Zugstabes.

```
public double BerechneErforderlichenQuerschnitt(double beanspruchung,
                                        String holzSortierklasse)
```
Methode zum Berechnen des erforderlichen Querschnitts des Zugstabes.

```
private double BestimmeSigma(String holzSortierklasse)
```
Den Sigma-Wert zu dieser Sortierklasse holen.

### Die Klasse *HolzDruckstab*

Die Klasse ist ebenfalls von *FunktionsObjekt* abgeleitet. Aufgrund der Übersichtlichkeit werden in Bild 102 die Attribute nicht angezeigt, jedoch nachfolgend genannt.




Für die Holznachweise auf Druck wurden insgesamt 4 unterschiedliche Methoden programmiert. Hierbei wurde auf Gleichungen aus Kapitel 2.7.1.1 verwendet, die im folgenden aufgelistet sind.

#### Normalkraftbeanspruchung auf Druck

Alle 4 Methoden basieren auf den Gleichungen Gl. (148) und Gl. (149).

Bild 102. UML Darstellung der Klasse *HolzDruckstab*

### Attribute

```
private int mConstantStatusOK = 0;              // alles Ok
private int mConstantStatusSchlank = 301;       // Schlankheitsgrad unzulässig
private int mConstantStatusDruck = 302;         // Nachweis auf Druck >1 (unzulässig)
private int mConstantStatusDatei = 303;         // Datei zum Einlesen nicht gefunden
private int mConstantStatusError = 304;         // Allgemeiner Fehler
private Liste mListe = new Liste();             // Ein Objekt der Klasse Liste
// Variablen, die an die Methoden übergeben werden
private double mBeanspruchung;                  // die einwirkende Kraft auf den Stab in kN
private double mKnicklaenge_y;                  // in cm
private double mKnicklaenge_z;                  // in cm
private String mSortierklasse;                  // Wert muss aus einer Tabelle ausgelesen werden
private double mQuerschnitt_b;                  // Breite des Querschnittes     in cm
private double mQuerschnitt_h;                  // Höhe des Querschnittes in cm
// Variablen, die intern benötigt werden
private double mSigma_parallel;                 // in kN/cm ^ 2
private double mQuerschnitt_A;                  // "ungeschwächter Querschnitt" in cm ^ 2
private double mTraegheitsradius_y;             // zur Berechnung des Schlankheitsgrades
private double mTraegheitsradius_z;             // zur Berechnung des Schlankheitsgrades
private double mTraegheitsvalue = 0.289;        // feste Variable
private double mSchlankgrad_y;                  // Kommentar folgt später
private double mSchlankgrad_z;                  // Kommentar folgt später
private double mSchlankgrad_mass;               // Kommentar folgt später
private double mSchlankgrad_grenz = 150;        // Kommentar folgt später
private double mKnickzahl;                      // wird anhand von mSchlankheitsgrad bestimmt
private double mKnick_OK;                       // Zulässige Knickspannung
private double mQuer_b;                         // Tempvariable für die Bemessung
private int mStatusSigma;                       // Flag, ob Sigma_parallel zulässig
private int mStatus;                            // Flag, ob Schlankheitsgrad zulässig
private int mSchlankOK;                         // Vergleichsvariable
private int i = 0;                              // Zählvariable als Abbruchkriterium bei
Bemessung
// Die Ergebnisse der jeweiligen Funktionen
private double mDrucknachweis;          // Ergebnis der Methode BerechneHolzDruckNachweis
private double mDruckkraft;             // Ergebnis der Methode BerechneHolzDruckKraft
private double mDruckbem;               // Ergebnis der Methode BerechneHolzDruckBem
```

### Methoden

Konstruktor

```
public HolzDruckstab()

public double HolzDruck_Nachweis(double setBeanspruchung, double
                        setKnicklaenge_y, double setKnicklaenge_z, String
                        setSortierklasse, double setQuerschnitt_b, double
                        setQuerschnitt_h)
```

Berechnung für den Drucknachweis. Ermittelung, ob der Schlankheitsgrad zulässig ist; wenn ja, wird der Drucknachweis geführt und geprüft, ob er das Spannungsverhältnis ≤ 1 ist. Ist das

der Fall wird das Spannungsverhältnis als Ergebnis zurückgeliefert, anderenfalls wird der jeweilige Fehlerstatus zurückgeliefert.

```
public double HolzDruck_zul_N(double setKnicklaenge_y, double
                              setKnicklaenge_z, String setSortierklasse,
                              double setQuerschnitt_b, double setQuerschnitt_h)
```

Berechnung der Holzdruckkraft. Ermittlung, ob der Schlankheitsgrad zulässig ist: wenn ja, wird die zulässige Kraft berechnet und zurückgeliefert, wenn nein, muss ein neuer Querschnitt gewählt werden.

```
public double HolzDruck_Bem(double setBeanspruchung,
                            double setKnicklaenge_y, double setKnicklaenge_z,
                            String setSortierklasse, double setQuerschnitt_h)
```

Bemessung für Druckkräfte:
Zunächst wird *mSigma_parallel* bestimmt und damit die Querschnittsbreite näherungsweise berechnet. Danach wird geprüft, ob der Schlankheitsgrad zulässig ist. Ist dies nicht der Fall wird der Querschnitt in Schritten von 2cm automatisch neu gewählt.

```
private int ErmittleSchlankheitsgradNachweisKraft()
```

In dieser Methode wird geprüft, ob der Schlankheitsgrad zulässig ist. Im Erfolgsfall wird der Status 0 zurückgeliefert, ansonsten der Status –1001. Sie wird von den Methoden *HolzDruckNachweis* und *HolzDruckKraft* benutzt. Die Methode *HolzDruckBemessung* benötigt eine eigene Methode.

```
private int ErmittleSchlankheitsgradBemessung(double tQuer_b)
```

Diese Methode wird von der Methode *HolzDruckBemessung* verwendet.
In dieser Methode wird geprüft, ob der Schlankheitsgrad zulässig ist. Im Erfolgsfall wird das Druckspannungsverhältnis errechnet. Ist dieser Wert ≤ 1 war die Berechnung erfolgreich, und der Status wird auf "OK" gesetzt. Im anderen Fall wird der Status auf "Schlank" gesetzt, was einen erneuten Aufruf der Methode mit einem modifizierten Wert zufolge hat.

```
private int ErmittleD_parallel(String Sortierklasse)
```

In dieser Methode wird Sigma_parallel ermittelt. Der Wert wird anhand des übergebenden Strings aus einer Datei eingelesen. Wird die Datei nicht gefunden, so wird ein Fehlerstatus zurückgegeben.

```
private double ErmittleQuerschnitt(double querschnitt_b, double querschnitt_h)
```

Gibt die Querschnittsfläche aus der Ermittlung von Breite x Höhe zurück.

```
private double ErmittleTraegheitsradius(double querschnitt, double wert)
```

Gibt den Trägheitsradius zurück, indem der Querschnitt mit dem Wert multipliziert wird.

```
private double ErmittleSchlankheitsMass(double klaenge_y, double tradius_y,
double klaenge_z, double tradius_z)
```

Die Methode ermittelt den maßgebenden Wert für die Schlankheit (Schlankheitsgrad).

```
private int VergleichSchlankheitsGrad(double grad, double grenz)
```

Die Methode überprüft, ob der ermittelte Schlankheitsgrad zulässig ist.

```
private double ErmittleKnickzahl(double schlankgrad)
```

Die Methode ermittelt den omega-Wert aus dem Schlankheitsgrad lamda.

private double ErmittleKnickspannung(double sigmaparallel, double knickzahl)
Die Methode dividiert sigmaparallel durch die knickzahl.

### Die Klasse *Liste*

Die Klasse *Liste* dient zum Einlesen einer Datei. Die abgespeicherten Werte können dann gezielt ausgelesen werden.




Bild 103. UML Darstellung der Klasse *Liste*

### Attribute

private Vector mBezeichnungen;
Im Vector *mBezeichnungen* werden die in der Datei hinterlegten Zeichenketten abgelegt. Das Format entspricht dem wie in der Datei (Groß- und Kleinschreibung), Leerzeichen am Anfang oder Ende werden entfernt.

private Vector mBezeichnungenKlein;
Im Vector *mBezeichnungenKlein* werden die in der Datei hinterlegten Zeichen abgelegt. Die in der Datei hinterlegte Zeichenkette wird in Kleinbuchstaben umgewandelt, Leerzeichen am Anfang oder Ende werden entfernt.

private Vector mWerte;
Im Vector *mWerte* werden die Zifferfolgen aus der Datei als Datentyp Double abgelegt. Die Werte können den Bezeichnern über den Index zugeordnet werden

### Methoden

Konstruktor
public Liste()

public void EinlesenListe(String dateiName)
Die Methode liest die Datei mit dem *dateiName* ein. Im Parameter *dateiName* muss der Dateiname mit dem absoluten Pfad angegeben werden. Das Zeichen "\" muß als "\\" übergeben werden, da es sonst als ESC Zeichen interpretiert wird. Bsp.: "C:\\temp\\Stahlguete.txt". Bei den Dateien handelt es sich um eigene geschriebene Dateien, welche Werte für Profile oder beispielsweise Holzgüten enthalten. Der Aufbau der Datei wird im Klassenkopfkommentar im Quellcode beschrieben.
public double GetWert(String bezeichnung)

Die Methode liefert den Wert der zu der im String *bezeichnung* übergebenen Zeichenkette gehört Groß- und Kleinschreibung wird beim Vergleich nicht berücksichtigt.

public String GetBezeichnungKleiner(double wert)
Die Methode gibt die nächst kleinere Bezeichnung zurück, d.h. der Tabellenwert der zurückgelieferten Bezeichnung ist der nächstkleinere in Bezug auf den übergebenen Wert.

public String GetBezeichnungGroesser(double wert)
Die Methode gibt die nächst höhere Bezeichnung zurück, d.h. der Tabellenwert der zurückgelieferten Bezeichnung ist der nächstgrößere in Bezug auf den übergebenen Wert

private boolean DoubleKompatibel(String testString)
Die Methode überprüft, ob die durch *testString* übergebene Zeichenkette in einen gültigen Doublewert umgewandelt werden kann, d.h. es darf max. ein '.' und sonst nur Zahlen als Zeichen vorkommen. Kann der String in ein Doublewert umgewandelt werden, wird *true* zurückgeliefert, ansonsten *false*.

private String GetDLLRegistryPath()
Diese Methode liest den Pfad aus der Windows-Registry unter dem die DLL installiert wurde. Das macht das Lesen der Tabellendateien unabhängig vom Ort der Installation. Der aktuelle Schlüssel lautet // 04A335C6-EAD9-11D4-8AC0-00104B471BFE.

### 2.7.2.2 Die Bibliothek HolzDruckZugStab.dll

Die Bibliothek *HolzDruckZugStab.dll* beinhaltet die Berechnungen für Biegung mit Zug und Biegung mit Druck von Balken aus Holz. Die erforderlichen Stabilitätsnachweise sind darin integriert. Bild 104 zeigt das Klassendiagramm der Bibliothek.

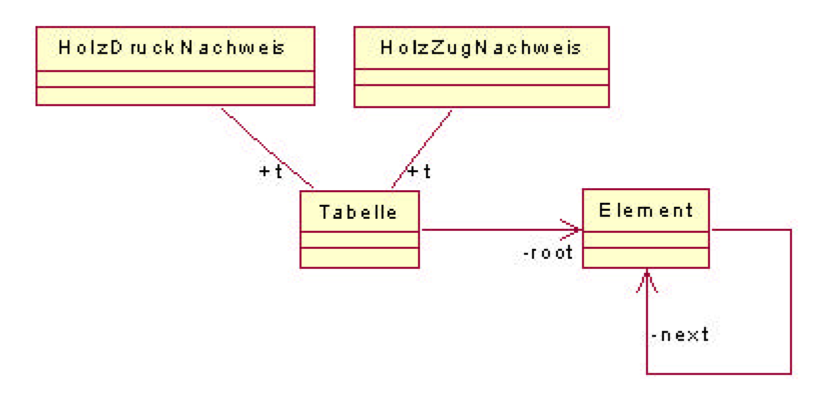



Bild 104. UML Klassendiagramm von *HolzDruckZugStab.dll*

Die Java Klassen lassen sich in zwei Gruppen aufteilen die Internen Klassen, welche hauptsächlich für die Verwaltung der Daten verantwortlich sind und die COM-Klassen, welche die Schnittstelle für die Objekterzeugung bereitstellen.

#### 2.7.2.2.1 Interne Klassen

Der Modul beinhaltet zwei Klassen die für die Nutzung von der COM–Ebene gesperrt sind. Das sind die Klassen Tabelle und Element. Diese zwei Klassen ermöglichen den Zugriff auf die Daten der in den Textdateien gespeicherten Tabellen.

### Die Klasse *Tabelle*

Die Klasse Tabelle Verwaltet die Daten aus einer Datei. Die Daten werden als eine verkettete Liste abgespeichert.

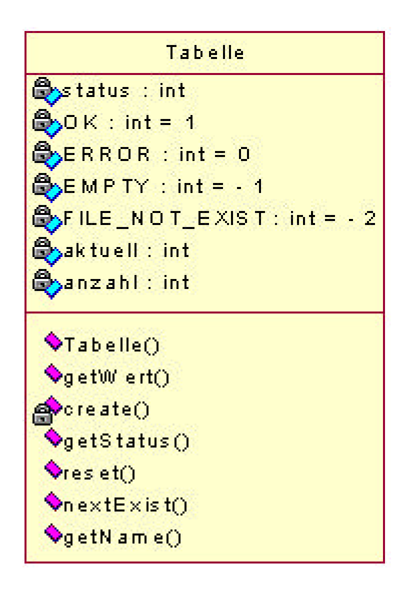



Bild 105. UML Darstellung der Klasse *Tabelle*

### Attribute

private int status // Zeigt den Status der Klasse an

private int OK = 1 // Konstanten Wert für den Status

private int ERROR = 0 // Konstante für Fehleranzeige

private int EMPTY = -1 // Konst. für fehlende Werte

private int FILE_NOT_EXIST = -2 // Konst. für fehlende Textdatei

private int aktuell // Interne Zähler für die Ausgabe der Überschrift. Nummer des
//aktuellen Elements in der Liste

private int anzahl // Anzahl der Elemente

private Element root // Der erste Element in der Verketteten Liste

### Methoden

public Tabelle(String path)
Konstruktor der Klasse *Tabelle*. Der Parameter *path* beinhaltet den absoluten Pfad zu der Datei, die eingelesen werden soll. Im Konstruktor werden die einzelnen Zeilen der Textdateien in einer verketteten Liste in je ein Objekt vom Typ *Element* eingelesen.

public double getWert(String zeile, int spalte)
Mit *Zeile* wird ein Schlüssel übergeben, mit dem auf die korrekte Zeile in der Tabelle zugegriffen werden soll. *Spalte* beinhaltet den Spaltenindex des Werts, der ausgelesen werden soll. Die Methode gibt ein Wert aus der Tabelle zurück.

private Element create(String zeile)
Gibt eine leere Instanz der Klasse *Element* zurück, bzw. generiert eine neue Instanz der Klasse.

public int getStatus()
Gibt Status der Klasse zurück.

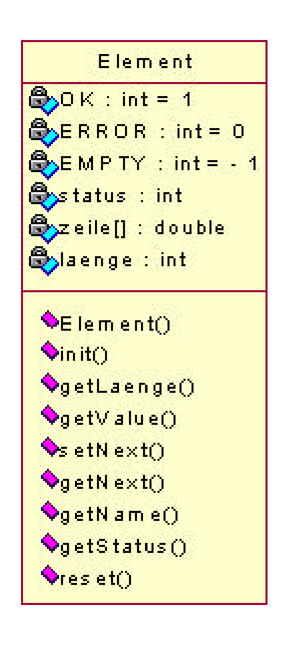
public void reset()
Setzt den Zeilenzähler *aktuell* auf 0, dadurch beginnt die Ausgabe der Zeilenüberschrift bei der ersten Zeile.

public int nextExist()
Die Methode gibt *OK* zurück, falls eine weitere Zeile existiert.

public String getName()
Die Methode gibt den Namen der aktuellen Spalte zurück.

## Die Klasse *Element*

Die Klasse Element verwaltet eine Zeile aus der Wertetabelle.




**Attribute**

private int OK = 1 // Konstanten Wert für den Status

private int ERROR = 0 // Konst. für das Auftreten eines Fehlers

private int EMPTY = -1 // Konst. für fehlende Werte

private int status // Status der Klasse

private Element next // Var. für das nächste Element

private String name // Name der Spalte

private double[] zeile // Die Werte aus den Zeilen der Textdateien

private int laenge // Anzahl der Zeichen der Zeile

Bild 106. UML Darstellung der Klasse *Tabelle*

**Methoden**

public Element()
Default Konstruktor. Hier werden bei der Objekterzeugung verschiedene Variablen, wie beispielsweise *status* mit Werten belegt

public int init(String zeileStr)
In der Methode *init* werden die unterschiedlichen Werte einer Zeile, die mit *zeileStr* als String übergenben wird, in ein Array vom Typ Double in einer bestimmten Reihenfolge eingelesen. Der Parameter *zeileStr* beinhaltet die Überschrift und die Werte der Zeile, die gelesen werden sollen. Mit Überschrift der Zeile ist die erste Zeichenfolge gemeint, wie beispielsweise NHS13.

public int getLaenge(){return laenge;}
Die Methode gibt die Länge des Array mit Werten zurück.

public double getValue(int spalte)
Die Methode gibt den Wert aus einer bestimmten Spalte des Array mit den Ergebniswerten zurück. Der Zähler beginn bei 1.

public int setNext(Element nextElem)
Die Methode speichert das nächste Element.

public Element getNext()
Die Methode gibt das nächste Element zurück

public String getName()
Die Methode gibt die Überschrift der Zeile zurück.

public int getStatus()
Die Methode gibt den Status der Klasse zurück.

public int reset()
Die Methode setz die Werte, die in der Klasse gespeichert waren zurück.

### 2.7.2.2.2 COM-Klassen

In der Bibliothek *HolzDruckZugStab.dll* wurden zwei COM-Klassen implementiert. Die Klasse *HolzZugNachweis* und die Klasse *HolzDruckNachweis*. Die Klasse beinhalten die eigentlichen Berechnungsroutinen für die unterschiedlichen Nachweise, die von außen über die COM-Technologie zur Verfügung stehen.

### Die Klasse HolzZugNachweis

HolzZugNachweis

status : int
OK : int = 1
ERROR : int = 0
EMPTY : int = -1
FILE_NOT_EXIST : int = -2
SCHLANKHEITSGRAD_NICHT_ZULAESSIG : int = -3
NACHWEIS_NICHT_ERFUELLT : int = -4

holzZugMyMz()
holzZug()
erforderlicheBreiteMyMz()
erforderlicheBreite()
sigmaParallel()
sigmaB()
GetDLLRegistryPath()
getStatus()

Bild 107. UML Darstellung der Klasse *HolzZugNachweis*

Für die Holznachweise auf Biegung mit Zug wurden insgesamt 6 unterschiedliche Methoden programmiert. Hierbei wurde auf Gleichungen aus Kapitel 2.7.1.2 verwendet, die im folgenden aufgelistet sind.

### Nachweise auf Biegung mit Zug

Alle 6 Methoden basieren auf Gleichung Gl. (152).

### Attribute

private int status // Status der Klasse

private int OK = 1 // Konstanten Wert für den Status

private int ERROR = 0 // Konst. für das Auftreten eines Fehlers

private int EMPTY = -1 // Konst. für fehlende Werte

private int FILE_NOT_EXIST = -2 // Konst. für fehlende Textdatei

private int SCHLANKHEITSGRAD_NICHT_ZULAESSIG = -3 // Konst. für nicht erfüllten Schlankheitsgrad

private int NACHWEIS_NICHT_ERFUELLT = -4 // Für Meldung, dass Nachweis nicht erfüllt ist

### Methoden

public double holzZugMyMz(double My,double Mz,double N,String HolzSortKlasse, double b, double h)
Die Methode führt den Nachweis eines auf Biegung in zwei Achsen und Zug beanspruchten rechteckigen Holzquerschnitts. Die Übergabeparameter sind in dieser Reihenfolge: das Bemessungsmoment $M_y$, das Bemessungsmoment $M_z$, die positive Normalkraft $N$, die Holzsortierklasse *HolzSortKlasse* als String, die Breite $b$ des Querschnitts und die Höhe $h$ des Querschnitts.

public double holzZug(double My,double N,String HolzSortKlasse, double b, double h)
Die Methode führt den Nachweis eines auf Biegung um die y-Achse und Zug beanspruchten rechteckigen Holzquerschnitts. Die Übergabeparameter entsprechen den Parametern in der Methode *holzZugMyMz*, mit dem fehlenden Moment $M_z$. In der Methode werden die Übergabeparameter an die Methode *holzZugMyMz* weitergegeben, indem für den zweiten Wert 0 übergeben wird. Folgende Zeile zeigt die Vorgehensweise hierbei.

```
exitValue=this.holzZugMyMz(My,0,N,HolzSortKlasse,b,h);
```

public double erforderlicheBreiteMyMz(double My,double Mz,double N, String HolzSortKlasse,double h)
Die Methode ermittelt die erforderliche Breite eines auf Zug und zweiachsiger Biegung beanspruchten rechteckigen Holzquerschnitts. Hierbei wird in einer Schleife der Nachweis in der Methode *holzZugMyMz* so lange geführt, bis der Nachweis eingehalten ist. Die zugehörige Breite wird dann als Ergebniswert geliefert. Folgender Code veranschaulicht die Vorgehensweise.

```
{ double exitValue=0;                          // setzen der Ergebnisvar. auf 0
double b=0.01;                                 // Festlegung des Startwerts für die Breite b
int schalter=1;                                // den Schalter für die While Schleife auf 1 setzen
while(schalter==1){                            // Start der Wehile Schleife
this.holzZugMyMz( My,Mz,N,HolzSortKlasse,b,h); // führt den Nachweis auf Biegung mit Zug
          if(status==OK){                                // wenn Berechnung zu einem sinnvollen
Ergebnis führte
                    exitValue=b;               // Festlegung des Ergebniswerts
                    break;                     // Beenden der While Schleife
          }                                    // Ende if
          b+=0.01;                             // Erhöhen der Breite um einen Zentimeter
     }                                         // Ende While
     return exitValue;                         // Rückgabe des Ergebnisses
     }                                         // Ende holzZugMyMz
```

Programmlisting 7.

public double erforderlicheBreite(double My,double N, String HolzSortKlasse,double h){
Die Methode ermittelt die erforderliche Breite eines auf Zug und einachsiger Biegung beanspruchten rechteckigen Holzquerschnitts. Hierbei werden die Werte an die Methode *erforderlicheBreiteMyMz* weitergegeben, indem der Wert für das Moment um die Z-Achse wieder auf 0 gesetzt wird.

public double sigmaParallel(String sortierklasse)
Die Methode liest den Wert für die zulässige Spannung für die Belastung parallel zur Faser für eine bestimmte Holzsortierklasse aus der Datei *HolzsortierklassenZug.txt* ein. Übergabeparameter ist ein String mit der gewünschten Holzsortierklasse.

public double sigmaB(String sortierklasse)
Die Methode liest den Wert für die zulässige Biegesspannung für eine bestimmte Holzsortierklasse aus der Datei *HolzsortierklassenSigmaB.txt* ein. Übergabeparameter ist ein String mit der gewünschten Holzsortierklasse.

private String GetDLLRegistryPath()
Diese Methode liest den Pfad aus der Windows-Registry unter dem die DLL installiert wurde. Das macht das Lesen der Tabellendateien unabhängig vom Ort der Installation. Der aktuelle Schlüssel lautet // 9A9D8D46-F67F-11D4-A4E1-006008A051C4

public int getStatus(){return status;}
Die Methode gibt den Status der Klasse als Rückgabeparameter an die aufrufende Stelle zurück.

### Die Klasse *HolzDruckNachweis*

Für die Holznachweise auf Biegung mit Druck wurden insgesamt 4 unterschiedliche Methoden programmiert. Hierbei wurde auf Gleichungen aus Kapitel 2.7.1.2 verwendet, die im folgenden aufgelistet sind.

### Nachweise auf Biegung mit Druck inklusive der Stabilitätsnachweise

Die 4 Methoden basieren auf Gleichung Gl. (153) bis Gl. (155).

### Attribute

```
private int status                                    // Status der Klasse
private int OK = 1                                    // Konstanten Wert für den Status
private int ERROR = 0                                 // Konst. für das Auftreten eines Fehlers
private int EMPTY = -1                                // Konst. für fehlende Werte
private int FILE_NOT_EXIST = -2                       // Konst. für fehlende Textdatei
private int SCHLANKHEITSGRAD_NICHT_ZULAESSIG = -3      // Konst. für nicht erfüllten Schlankheitsgrad
private int NACHWEIS_NICHT_ERFUELLT = -4              // Für Meldung, dass Nachweis nicht erfüllt ist
private double mTraegheitsValue = 0.289;              // Konst. für Trägheitsradius
private double mSchlankgrad_grenz = 250;              // Konst. für den Grenzschlankheitsgrad
private double gamma = 2;                             // Konst. für Ermittlung des Kippbeiwerts g l
```

HolzDruckNachweis

status : int
OK : int = 1
ERROR : int = 0
EMPTY : int = -1
FILE_NOT_EXIST : int = -2
SCHLANKHEITSGRAD_NICHT_ZULAESSIG : int = -3
NACHWEIS_NICHT_ERFUELLT : int = -4
mTraegheitsValue : double = 0.289
mSchlankgrad_grenz : double = 250
gamma : double = 2

holzDruckUndMyMz_Nachweis()
holzDruck_Nachweis()
sigmaB()
knickzahl()
Nr()
erforderlicheBreiteMyMz()
erforderlicheBreite()
sigmaParallelD()
E_Parallel()
Gt()
kb()
knickzahl()
schlankheitsgrad()
getStatus()
knickzahl()
GetDLLRegistryPath()

Bild 108. UML Darstellung der Klasse *HolzDruckNachweis*

## Methoden

public double holzDruckUndMyMz_Nachweis(double My,double Mz, double N, double s,double sKy,double sKz,
double h,double b, String HolzSortKlasse)

Die Methode führt den Nachweis eines auf Biegung in zwei Achsen und Druck beanspruchten rechteckigen Holzquerschnitts. Die Übergabeparameter sind in dieser Reihenfolge das Bemessungsmoment $M_y$, das Bemessungsmoment $M_z$, die negative Normalkraft $N$, die Knicklänge um die y-Achse *sky*, die Knicklänge um die z-Achse *skz*, die Höhe *h* des Querschnitts, die Breite *b* des Querschnitts und die Holzsortierklasse *HolzSortKlasse* als String.

public double holzDruck_Nachweis(double My, double N, double s,double sKy,double sKz,
double h,double b, String HolzSortKlasse)

Die Methode führt den Nachweis eines auf Biegung um die y-Achse und Druck beanspruchten rechteckigen Holzquerschnitts. Die Übergabeparameter entsprechen den Parametern in der Methode *holzZugMyMz*, mit dem fehlenden Moment $M_z$. In der Methode

werden die Übergabeparameter an die Methode *holzDruckUndMyMz_Nachweis* weitergegeben, indem für den zweiten Wert 0 übergeben wird.

private double sigmaB(String sortierklasse)
Die Methode liest den Wert für die zulässige Biegespannung für eine bestimmte Holzsortierklasse aus der Datei *HolzsortierklassenSigmaB.txt* ein. Übergabeparameter ist ein String mit der gewünschten Holzsortierklasse.

private double knickzahl(String sortierklasse,double lambda)
Die Methode ermittelt die Knickzahl *w* für den Nachweis auf Biegung mit Druck.

private int Nr(String sortierklasse){
Die Methode ermittelt die Spalte in der Datei *Knickzahlen.txt*, in der der zugehörige Wert *w* für eine Holzsortierklasse gespeichert ist.

public double erforderlicheBreiteMyMz(double My,double Mz, double N, double s,double sKy,double sKz, double h,String HolzSortKlasse)
Die Methode ermittelt die erforderliche Breite eines auf Druck und zweiachsiger Biegung beanspruchten rechteckigen Holzquerschnitts. Die Werte werden an die Methode *holzDruckUndMyMz_Nachweis* weitergegeben. Hierbei wird in einer Schleife der Nachweis in der Methode *holzDruckUndMyMz_Nachweis* so lange geführt, bis der Nachweis eingehalten ist. Die zugehörige Breite wird dann als Ergebniswert geliefert.

public double erforderlicheBreite(double My, double N, double s,double sKy,double sKz, double h,String HolzSortKlasse){
Die Methode ermittelt die erforderliche Breite eines auf Zug und einachsiger Biegung beanspruchten rechteckigen Holzquerschnitts. Hierbei werden die Werte an die Methode *erforderlicheBreiteMyMz* weitergegeben, indem der Wert für das Moment um die Z-Achse auf 0 gesetzt wird.

private double sigmaParallelD(String sortierklasse)
Die Methode ermittelt die zulässige Druckspannung parallel zu Fasern aus der Datei *HolzsortierklassenDruck.txt*

private double E_Parallel(String sortierklasse)
Die Methode ermittelt die zulässige Druckspannung parallel zu Fasern aus der Datei *HolzsortierklassenE.txt*

private double Gt(String sortierklasse)
Die Methode ermittelt die zulässige Druckspannung parallel zu Fasern aus der Datei *HolzsortierklassenGt.txt*

private double kB(double zulSigmaB,double Epar,double Gt)
Die Methode ermittelt den zulässigen Kippbeiwert $k_b$ zur Ermittlung des Kippschlankheitsgrads $l_B$.

private double knickzahl(double knicklaengeY,double knicklaengeZ, double b,double h)
Ermittelt über die Funktion schlankheitsgrad den maßgebenden Schlankheitsgrad *l* und überprüft, ob den Schlankheitsgrad zulässig ist.

private double schlankheitsgrad(double sKy,double sKz,double iy,double iz){
Ermittelt den maßgebenden Schlankheitsgrad *l* für den Spannungsnachweis.

public int getStatus(){return status;}
Die Methode gibt den Status der Klasse als Rückgabeparameter an die aufrufende Stelle zurück.

private double knickzahl(String name,String klass){
Die Methode ist überschrieben und ermittelt die Knickzahl *w* für den Nachweis auf Biegung mit Druck. Die Übergabeparameter sind unterschiedlich zu den vorangegangenen erklärten Methoden.

private String GetDLLRegistryPath()
Diese Methode liest den Pfad aus der Windows-Registry unter dem die DLL installiert wurde. Das macht das Lesen der Tabellendateien unabhängig vom Ort der Installation. Der aktuelle Schlüssel lautet // 9A9D8D46-F67F-11D4-A4E1-006008A051C4

## Literatur zu Kapitel II-2

[1] Falk S., „Die Berechnung des beliebig gestützten Durchlaufträgers nach dem Reduktionsverfahren“, Ingenieur-Archiv, Springer-Verlag Berlin, 1956

[2] Kersten, R., Das Reduktionsverfahren der Baustatik - Verfahren der Übertragungsmatrizen, 2.Auflage, Springer Verlag, 1982

[3] Werkle, H., Vorlesungsskript Bauinformatik II, Fachhochschule Konstanz

[4] H. Rubin, Statik ebener Stabwerke, Stahlbau Handbuch, Band 1, Teil A, Stahlbau-Verlagsgesellschaft mbH, Köln 1993

[5] Werkle H., Finite Elemente in der Baustatik, 2. Auflage, Vieweg, Wiesbaden, 2001

[6] Edatra I, Studentisches Projekt im Fachbereich Informatik der Fachhochschule Konstanz, WS 2000/2001, Betreuer: Prof. Dr. H. Pleßke, Prof. Dr.-Ing. H. Werkle

[7] Link: http://www.ee.oulu.fi/~timor/javaa

[8] Meißner, Maurial, Die Methode der Finiten Elemente, Eine Einführung in die Grundlagen, 2. Auflage, Springer Verlag, 2000

[9] Schneider, Bautabellen für Ingenieure, 14. Auflage, Werner Verlag, Düsseldorf, 2001

[10] Schneider, Bautabellen für Ingenieure, 13. Auflage, Werner Verlag, Düsseldorf, 1998

[11] Deutscher Ausschuss für Stahlbeton, DAfStb Heft 425, Bemessungshilfen zum EC 2 Beuth-Verlag, 1992

[12] Brugger K., Günther J., Softwareentwicklung zur Anwendung der Relationalen Datenbank MS-ACCESS im konstruktiven Ingenieurbau, Diplomarbeit, Fachhochshcule Konstanz, 2001, Betreuer: Prof. Dr.-Ing. H. Werkle

[13] Schneider, Bautabellen für Ingenieure, 11. Auflage, Werner Verlag, 1994

[14] EUROPROFIL S.A., Societe commerciale de PROFILARBED
66, rue de Luxembourg, 4009 Esch-sur-/Alzette, Luxembourg

# II-3 Benutzeroberflächen

## 3.1 Allgemeines

Dieses Kapitel beschäftigt sich hauptsächlich mit der programmtechnischen Umsetzung der Oberflächen. Ein Handbuch für die Benutzung der verschiedenen Oberflächen wurde ebenfalls erstellt und ist im folgenden Kapitel II-4, Handbücher für die Softwarebenutzzung, zu finden.

Im folgenden sind die verschiedenen Formmodule der Anwendung erläutert. Sie bilden die eigentliche Schnittstelle zwischen Anwender und Anwendung. Über sie kommuniziert der Anwender mit dem Programm. Je strukturierter und übersichtlicher ihr Aufbau ist und um so mehr Informationen für den Benutzer bereitgestellt werden, um so besser funktioniert der Datenaustausch und um so weniger Probleme treten während des Programmablaufs auf.

Für die unterschiedlichen Anwendungen ergaben sich verschiedene Gestaltungsmöglichkeiten für die Eingabeoberflächen. Wobei sich für die Anbindung des Softwarebaukastens an die Office Anwendungen die meisten Freiheiten ergaben.

## 3.2 Die Oberflächen in MS-Excel

### 3.2.1 Die Standardeingabemöglichkeit in Excel

Diese Möglichkeit der Eingabe stellt die einfachste Art der Bereitstellung einer Benutzeroberfläche dar. Hierauf, sowie auf deren Vor- und Nachteile wurde jedoch bereits in Kapitel 1.4.1 eingegangen. So kann an dieser Stelle auf eine nochmalige genaue Beschreibung der Vor-, Nachteile und Vorgehensweise bei der Bereitstellung der Funktionen verzichtet werden. Die Anbindung wird nachfolgend jedoch anhand eines Codebeispiels für die Funktion *Function HolzZugM_erfBreite*[1] erläutert.

Zu beachten gilt noch, dass die Bereitstellung der Funktionen, bzw. der VBA Code hierfür innerhalb der Office Version 2000 erstellt wurden. Dies kann bei älteren Versionen von Office unter Umständen zu Fehlermeldungen führen, falls Schlüsselwörter in VBA bei der Programmierung verwendet wurden, die in älteren Versionen der Sprache noch nicht verfügbar waren.

In Excel wird auf die Funktionen über den Menübefehl *Einfügen/Funktion* zugegriffen. Bild 26 in Kapitel 1.4.1.1 zeigt das Dialogfenster.

Nach der Bestätigung wird das Standardeingabefenster für die Funktion geöffnet (Bild 109). Das Fenster ist auf die programmierte Funktion zugeschnitten und wird von MS-Excel automatisch generiert, sobald ein VBA Macro existiert, das eine entsprechende Funktionsdeklaration besitzt, wie Zeile 1: des folgenden Programmlisting zeigt. In diesem Fall werden vier Eingabewerte benötigt.

[1] Die Funktion liefert die erforderliche Breite in cm eines Biegezugstabes aus Holz mit einachsiger Biegung

HolzZugM_erfBreite
My_kNm =
N_kN =
H_m =
Sortierklasse =
=
Klicken Sie auf die Schaltfläche 'Hilfe', um weitere Hilfe für diese Funktion und die Argumente zu bekommen.
My_kNm
Formelergebnis =
OK Abbrechen

Bild 109. Von Excel erzeugtes Fenster für die Eingabe von Werten

Zeile 1: *Function HolzZugM_erfBreite(My_kNm As Double, N_kN As Double, _*
*h_m As_Double, sortierklasse As String) As Double*

Die Eingabeparameter stehen in Klammer nach dem Funktionsaufruf. Man spricht hier auch von Übergabeparametern, welche die Funktion erwartet. Anschließend folgt der Deklarationsteil der Routine.

Text im Programmcode, der hinter einem „'" Zeichen steht wird vom Programm ignoriert und auch bei der Kompilierung nicht berücksichtigt und kann deshalb im Code verbleiben. Mit *Dim* werden Variablen deklariert, die nur innerhalb eines Moduls oder einer Prozedur gültig sind. Wird die Prozedur beendet, wird die Variable zurückgesetzt, was auch Speicherplatz spart.

In Zeile 3: wird eine Objektvariable deklariert, die später eine Referenz auf eine in Java programmierte COM-Klasse oder COM-Objekt vom Typ *HolzZugNachweis* erhalten soll. *HolzZugNachweis* ist eine mit *Public* deklarierte Klasse im Java Programmcode, die damit öffentlich ist und auch vor dem Compilieren als COM-Klasse deklariert werden muss. Die Vorgehensweise hierzu wurde in Kapitel 1.3.1.1 näher erläutert. *HolzZugNachweis* wiederum ist ein Teil der Bibliothek *HolzDruckZugStab.dll*, in welcher innerhalb des Projekts die Funktionen für Nachweise im Holzbau gesammelt sind. Hier ist zu beachten, dass an dieser Stelle erst Speicherplatz besorgt wird, also noch kein Objekt instanziert wurde. Dies wird erst in Zeile 4: erledigt. Hierzu wird das Schlüsselwort *Set* in Verbindung mit *new* benötigt. Diese Zeilen verdeutlichen anschaulich, wie einfach die Erzeugung und Nutzung eines Objekts mittels COM-Technologie ist.

Zeile 2: *Dim ExitValue As Double* *' Var. für den Ausgabewert*

Zeile 3: *Dim myRef As HolzZugNachweis* *' Objektvariable*

Zeile 4: *Set myRef = New HolzZugNachweis*

Zeile 5: *ExitValue = myRef.erforderlicheBreite(My_kNm, N_kN, sortierklasse, h_m)*

Zeile 6: *If (myRef.getStatus < > 1 And myRef.getStatus < > 2) Then*

Zeile 7: *errorhandling (myRef.getStatus)*

Zeile 8: *End If*

‘Ausgabe des Ergebniswerts

Zeile 9: *HolzZugM_erfBreite = ExitValue*

Zeile 10: *End Function* *’ Ende HolzZugM_erfBreite*

Programmlisting 8.

Bild 110 zeigt nochmals den Projekt Explorer von Visual Basic für die Excel Anwendung. Beim Start von MS-Excel werden alle Dateien, die im XLStart Verzeichnis von Office liegen in den Arbeitsspeicher geladen und sind damit auch im Projekt Explorer sichtbar, wie an den vier AddIns mit der Dateiendung .xla ersichtlich ist. Wie bereits erwähnt sind Dateien mit der Endung .xla jedoch lediglich über den Visual Basic Editor bearbeitbar. Deren Arbeitsblätter sind in der Anwendung selbst, ohne Änderung der *IsAddin*-Eigenschaft der Arbeitsmappe nicht für Eingaben verfügbar.

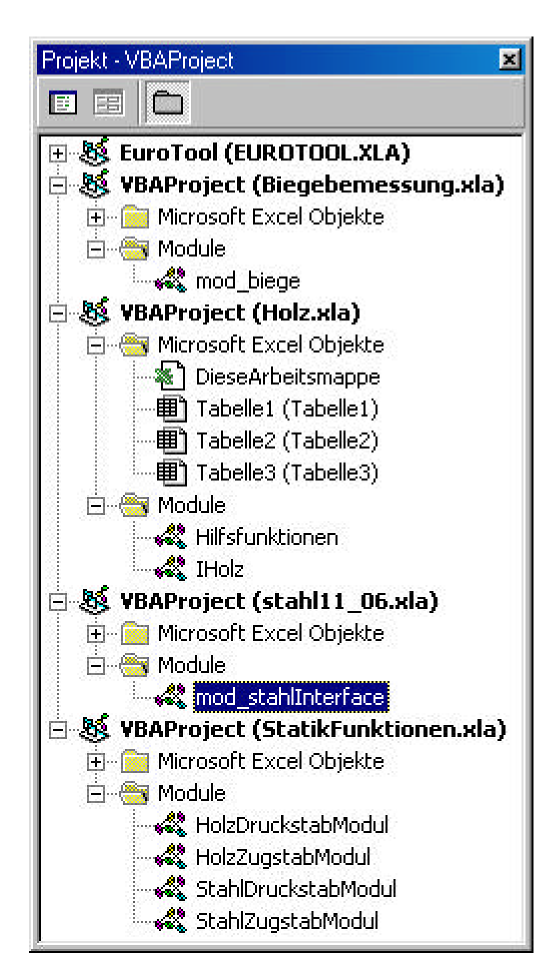



Bild 110. Projektexplorer in MS-Excel

### 3.2.2 Menüleiste für ED_TRA Funktionen

Nach der Installation des ED_TRA Softwarebaukastens liegt das AddIn *stahl11_06.xla*, wie in Bild 110 angezeigt, im XLStart Verzeichnis von MS-Excel. In ihr ist innerhalb des Moduls *mod_stahlInterface* in der Funktion *menueProfile* die Menüleiste der ED_TRA Funktionen als

VBA Code definiert. Die Routine wird beim Laden der Datei ausgeführt. Hierzu wurde in der Arbeitsmappe der Datei folgender Programmcode hinzugefügt:

```
Zeile 1:   Private Sub Workbook_Open()
Zeile 2:   menueProfile            ' Initialisiert die Menüleiste für die ED_TRA Funktionen
Zeile 3:   End Sub                 ‘ Ende Workbook_Open
```

Programmlisting 9.

Nach dem Start von *menueProfile* wird mit dem With-Block[2] von Zeile 2: bis Zeile 4: ein Trennstrich in im sechsten Menue der Menüleiste von MS-Excel hinzugefügt.

```
Zeile 1:   Sub menueProfile()                          ‘ Start menueProfile
Zeile 2:   With MenuBars(xlWorksheet).Menus(6).MenuItems
Zeile 3:     Set strich1 = .Add(Caption:="-")
Zeile 4:   End With
```

In Zeile 5: wird das Menü ED_TRA-Funktionen im Menü eingefügt und im folgenden With-Block werden auch die Menübefehle für die unterschiedlichen Berechnungsfunktionen eingefügt. Durch ein Komma getrennt, kann mit dem Schlüsselwort *OnAction* definiert werden, was nach einem Klick auf den Befehl geschehen soll. Hier wird auf weitere Funktionen verwiesen, welche die Eingabeoberflächen in den Speicher laden. Dies wird unten noch näher erläutert.

```
Zeile 5:     Set nm = MenuBars(xlWorksheet).Menus(6).MenuItems.AddMenu("ED_TRA-&Funktionen")
Zeile 6:   With nm.MenuItems
Zeile 7:     .Add Caption:="Profiltafeln", OnAction:="tafelWerte"
Zeile 8:     .Add Caption:="kh-Verfahren", OnAction:="funktion_Kh"
Zeile 9:     .Add Caption:="Durchlaufträger", OnAction:="funktion_Dlt"
Zeile 10:  End With                                    ‘ Ende With-Block
Zeile 11:  End Sub                                     ‘Ende menueProfile
```

Programmlisting 10.

[2] Mit *With* und dem dahinterstehenden Namen eines Objekts, hier die Menüleiste von MS-Excel, wird ermöglicht innerhalb des With-Blocks direkt auf Eigenschaften des Objekts mit einem vorangestellten Punkt zuzugreifen

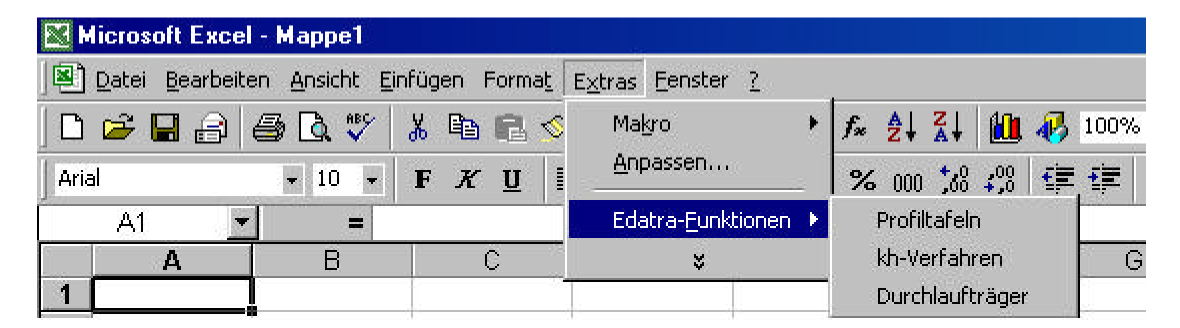



Bild 111. Menübefehl in MS-Excel

So wird nach dem öffnen des AddIn die Menueleiste mit dem Ausführen des Codes der Routine *menueProfile* initiert und die Funktionen des Softwarebaukastens, für die eigene Oberflächen programmiert wurden sind in Excel verfügbar, wie Bild 111 zeigt.

### 3.2.3 Eigene programmierte Oberflächen in dynamisch gelinkten Bibliotheken am Beispiel von edtra_DltVB

Die Gründe für die Programmierung, bzw. die Bereitstellung eigener Oberflächen wurde bereits in Kapitel 1.4 erläutert. Hierbei wurden auch die Eigenschaften von Servern und die Projekteinstellungen in der Entwicklungsumgebung von Visual Basic bei der Erstellung eines Projekts für eine dynamisch gelinkte Bibliothek dargelegt.

Die Programmierung einer Oberfläche ist sehr aufwendig. Beispielsweise hat allein die Eingabeoberfläche für die Berechnung von Durchlaufträgern mehr als 6500 Zeilen Code. Daher soll an dieser Stelle an diesem konkreten Beispiel das Erstellen eines eigenen Formulars aufgezeigt werden. Jedoch ist es aufgrund des eben erwähnten Umfangs hier nicht möglich, eine umfassende Erklärung zu allen Prozeduren, Funktionen, Eigenschaften, Methoden und Klassen zu geben. Für den weiter Interessierten bietet sich jedoch die Möglichkeit den Quellcode näher zu betrachten, da dieser zu allen wichtigen Teilen des gesamten Programms sowie zu allen Prozeduren, Deklarationen, Klassen und dergleichen Erklärungen bietet. Weiterhin wird in diesem Kapitel der grundsätzliche Aufbau der Software erläutert.

#### 3.2.3.1 Programmmodule

Jedes Visual Basic Programm ist im wesentlichen aus Formmodulen, allgemeinen Modulen und Klassenmodulen aufgebaut. Diese unterschiedlichen Module werden auf sinnvolle Weise eingesetzt und verknüpft, was einen klar gegliederten und strukturierten Aufbau der Software gewährleisten soll. Dies ermöglicht auch im Hinblick auf spätere Änderungen und Updates einen schnelleren Wiedereinstieg ins Programm. Im Grunde genommen sind Module jedoch nichts anderes als Visual Basic Dateien.

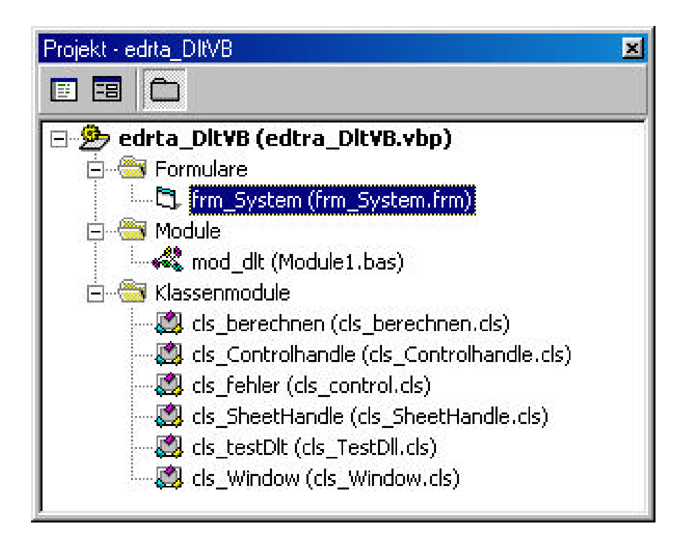



Bild 112. Projektstruktur von edtra_DltVB

Bild 112 zeigt die Projektstruktur , wie sie in der Entwicklungsumgebung von Visual Basic erscheint.

<<Class Module>>
cls_Window
<<Const>> HWND_TOPMOST = -1
<<Const>> HWND_NOTOPMOST = -2
<<Const>> SWP_NOSIZE = &H1
<<Const>> SWP_NOMOVE = &H2
<<Const>> SWP_NOACTIVATE = &H10
<<Const>> SWP_SHOWWINDOW = &H40
<<Const>> TOPMOST_FLAGS = SWP_NOMOVE Or SWP_NOSIZE Or SWP_NOACTIVATE
<<Const>> HWND_BOTTOM = 1
<<Const>> HWND_TOP = 0
MakeNormal()
MakeTopMost()
MakeBottom()
mutterFenster()

<<Class Module>>
cls_testDlt
fenster : frm_System
Test_dlt()

<<Class Module>>
cls_SheetHandle
frm()
GetExcel()
DetectExcel()
transformLager()

<<Module>>
mod_dlt
boolPunkt : Boolean
boolAbschnitt : Boolean
boolcontrol : Boolean
hoehe : Long
weite : Long
<<Declare>> FindWindow()
<<Declare>> SendMessage()
<<Declare>> SetParent()
<<Declare>> SetWindowPos()

+wb
+berechnen
+globalControl
-fenster

<<Form>>
frm_System
reiheOld_Punkt : Integer
reihe_punkt : Integer
reihe_Slast : Integer
reiheOld_Slast : Integer
reihe_Plast : Integer
reiheOld_Plast : Integer
reiheOld_Abschnitt : Integer
reihe_abschnitt : Integer
reihe_lager : Integer
reiheOld_lager : Integer
boolHighlight_lager : Boolean
boolHighlight_Slast : Boolean
boolHighlight_Plast : Boolean
boolHighlight_Punkt : Boolean
boolHighlight_abschnitt : Boolean
booldirty_lager : Boolean
booldirty_Slast : Boolean
booldirty_Plast : Boolean
booldirty_punkt : Boolean
booldirty_abschnitt : Boolean

<<Class Module>>
cls_berechnen
impact : Integer
solution : Double
überlagerung : Double
auflager : Double
auflagerÜberlagerung : Double
lastfallAnz : Integer
maxAbschnitt : Double
maxFeldmoment : Double
<<Let>> minMax()
<<Get>> minMax()
<<Let>> feldAbschnitte()
<<Get>> feldAbschnitte()
<<Let>> auflagerErgebniss()
<<Get>> auflagerErgebniss()
<<Let>> auflagerErgebnissÜberlagerung()
<<Get>> auflagerErgebnissÜberlagerung()
<<Let>> Lastfall()
<<Get>> Lastfall()
<<Let>> ergebnisse()
<<Get>> ergebnisse()
<<Let>> ergebnisseÜberlagerung()
<<Get>> ergebnisseÜberlagerung()
Übergabe()
getWerte()
leseLager()
leseAbschnitt()
transformLager()
ermittleLastfall()
sortLastfall()

<<Class Module>>
cls_Controlhandle
neuerAbschnitt : Integer
boolNeuerAbschnitt : Boolean
Feld_P : Variant
Feld_S : Variant
<<Let>> boolnewabschnitt()
<<Get>> boolnewabschnitt()
<<Let>> einfügenAbschnitt()
<<Get>> einfügenAbschnitt()
Wert()
schreibe_flex()
getLast()
sortieren()
zeigeLastgruppe()
leereFlex()
leereTextbox()
checkflex()
checkflexOld()
getTextbox()
textbox_lostfocus()
selectAbschnittslast()
prüfeLastfall()
prüfeFeld()
prüfeAbschnitt()
eintrag_löschen()
eintrag_einfügen()

<<Class Module>>
cls_fehler
flex_reihenTest()
textboxtest()
pruefeAbschnittslast()

1 1 1 1 1 1 1 0..1 1 1 1 1 1 1 1 1

Bild 113. UML Klassendiagramm von *edtra_DltVB*

In Bild 113 ist das Klassendiagramm des Projekts zu sehen. Im Diagramm wurden die 145 Steuerelemente, wie Befehlsschaltflächen, Anzeigefelder, Combo Boxen und dergleichen,

welche das Formmodul *frm_System* als Container enthält, der Übersichtlichkeit halber nicht angezeigt. Weiterhin sind die Methoden in *frm_System* ausgeblendet.

Insgesamt beinhaltet *edtra_DltVB.dll* ein Formmodul sowie ein allgemeines Modul und sechs Klassenmodule. Nachfolgend werden die verschiedenen Module näher erläutert, die wichtigsten Prozeduren und Methoden erklärt und es wird auf die Besonderheiten der Programmierung eingegangen.

Wie oben zu sehen ist, besteht auch eine DLL aus Formularmodulen mit der Endung *.frm*, allgemeinen Modulen mit *.bas* Endung und Klassenmodulen mit der Endung *.cls.* Zusätzlich erhält jedes dieser Module ein Präfix, was die Programmierung erleichtert und weiteren Überblick bringt. Der Vollständigkeit halber sind verwendeten Präfixe, welche der Konvention entsprechen auch in Anhang F genannt.

### 3.2.3.1.1 Objekte, Eigenschaften, Methoden und Ereignisse

**Objekte**

Von Objekten war bereits öfters die Rede. Die Klassenbibliothek *edtra_DltVB.dll* enthält verschiedene Klassen, deren Instanzen Objekte darstellen können. Beispielsweise wird *edtra_DltVB.dll* über die einzige öffentliche Klasse *cls_testDlt* angesprochen. Mit dem Aufruf wird ein eigenständiges Objekt der Klasse erzeugt. Wie in Kapitel 1.4.2 bereits angesprochen wurde die Instancing-Eigenschaft der Klasse *cls_testDlt* auf *InSameProcessMultiUse* gesetzt. Dies ermöglicht anderen Anwendungen das Erstellen von Objekten aus der Klasse. Eine Instanz des Servers der Komponente kann eine beliebige Anzahl von auf diese Weise erstellten Objekten bereitstellen. Multithreading ist hierbei nicht möglich. In der Klasse *cls_TestDlt* existiert wiederum nur eine einzige Methode, *Test_dlt(),* welche das Formular in den Speicher lädt, das selbst auch wieder ein Objekt darstellt. Genau betrachtet steckt hinter all dem wieder die COM-Technologie.

In Visual Basic werden verschiedene Klassen oder Objekte vom Typ Formular bereitgestellt. Diese besitzen je nach Anwendungsspektrum unterschiedliche Eigenschaften und Methoden, in denen sich jedoch auch viele Schlüsselwörter, Eigenschaften und Methoden entsprechen. Dies ist ein Vorteil im Umgang mit Objekten, da somit ein einheitliches Konzept für den Umgang mit Datenstrukturen zugrunde liegt.

**Eigenschaften**

Eigenschaften sind die charakteristischen Merkmale einer Klasse oder eines Objekt. So zum Beispiel die Farbgebung, Größe oder Position des Objekts auf dem Bildschirm. Die Eigenschaften werden in der Entwicklungsumgebung in einem eigenen Eigenschaftsfenster zum jeweils aktuellen Objekt bzw. zum Objekt, welches den Fokus besitzt angezeigt, wie Bild 114 für das Eigenschaftsfenster des Formulars *frm_System* zeigt.

Viele der Eigenschaften sind bereits in der Entwicklungsumgebung verfügbar und können bequem über das Eigenschaftsfenster eingestellt werden. Es bestehen jedoch auch weitere Eigenschaften, die im Programmcode angesprochen werden müssen. Dies geschieht dann formal immer mit der Nennung des Objekts und der durch einen Punkt getrennten und zugestellten Eigenschaft dahinter, wohinter sich wieder die COM-Technologie verbirgt.

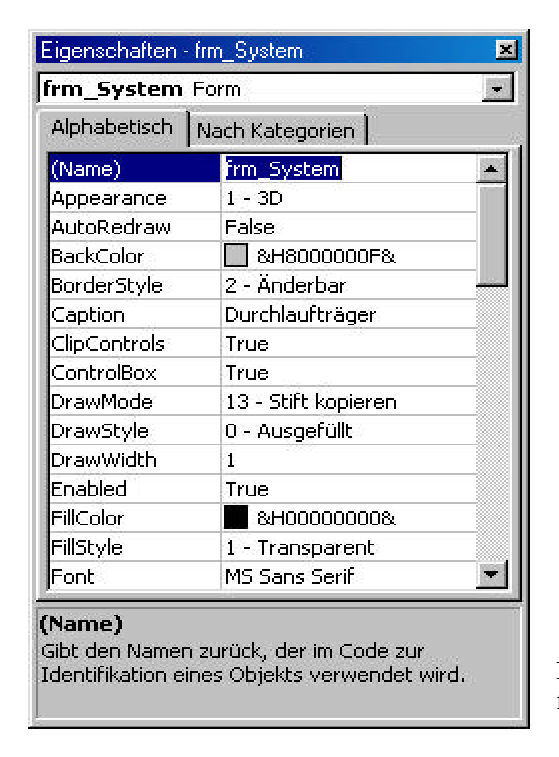



Bild 114. Eigenschaftsfenster für das Formular frm_System

### Methoden

Methoden führen Anweisungen aus und können eher als Prozeduren angesehen werden. Eine Methode eines Formulars wäre beispielsweise *show*, mit dem das Formular in den Speicher geladen und angezeigt wird. Angesprochen werden die Methoden wieder mit dem vorangestellten Objektnamen.

```
frm_System.show                                    'Anzeigen des Formulars
```

### Ereignisse

Ereignisse bei einem Formular sind zum Beispiel das Drücken einer Taste oder das Bewegen der Maus. Bild 115 zeigt einen Ausschnitt der Ereignisse des Formulars *frm_System*. Visual Basic bietet zu jedem Steuerelement, so auch zu Formularen, eine Auswahl an Ereignissen in einer Combo-Box[3] an, die durch Anklicken ins Codefenster übernommen werden können. Der Vorteil hierin ist, dass sich der Programmierer über die spezifischen Parameter in Klammern, an die beim

Resize
Activate
Click
DblClick
Deactivate
DragDrop
DragOver
GotFocus
Initialize
KeyDown
KeyPress
KeyUp
LinkClose

Bild 115. Ereignisse des Formulars frm_system

[3] Eine Combo-Box ist ein Steuerelement das eine Anzahl von Elementen enthält, die vom Benutzer ausgewählt werden können

Auftreten des Ereignisses Werte übergeben werden, keine Gedanken machen muss, da sie mitgeliefert werden. Nach dem Klicken auf das Ereignis erscheint im Codefenster der Methodenrumpf der Subroutine.

### 3.2.3.2 Formmodul

Im Folgenden wird das Formular der Anwendung erklärt. Es bildet die eigentliche Schnittstelle zwischen Anwender und Anwendung. Bild 116 zeigt das Formular, wie es in der Entwicklungsumgebung von Visual Basic erscheint. Das Formmodul bildet den Container für insgesamt weitere 145 Steuerelemente. Um die Größe der Anwendung und auch den Programmieraufwand innerhalb eines Forschungsprojekts in Grenzen zu halten wurde nur eine Form oder Fenster für die Eingabe verwendet, wie im unteren Bild zu sehen ist. Um dies zu erreichen, wurde die Oberfläche, die Art wie die Daten vom Benutzer einzugeben sind sowie die Aufbereitung der Daten für die Übergabe an die Berechnungskomponente *DLT.dll*, an diese angepasst. Die Bibilothek *DLT.dll* erwartet die Daten abschnittsweise von links oder vorn am Träger beginnend, bis rechts oder hinten durchlaufend. Abschnittsweise im Sinne des Durchlaufträgerprogramms bedeutet, dass der Benutzer vor der Eingabe des Systems die einzelnen Felder, sofern nötig, in einzelne Abschnitte aufteilen muss, was beispielsweise immer nötig ist, wenn eine einzelne Last innerhalb eines Feldes angreift oder eine Linienlast innerhalb eines Feldes beginnt oder endet.

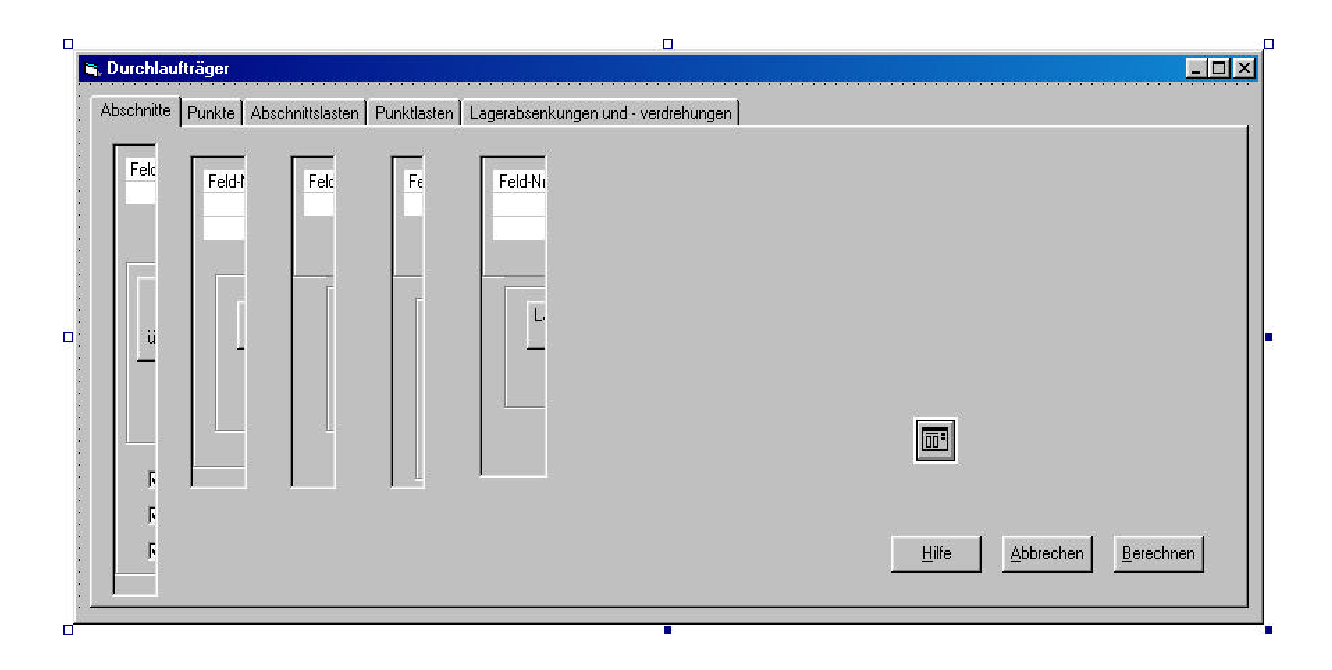



Bild 116. Das Eingabeformular frm_System für Durchlaufträgerberechnungen in MS-Excel

Auf dem Formular ist das Commondialog Steuerelement angeordnet. Das Steuerelement stellt die Schnittstelle zwischen VB und der *COMMDLG.DLL* von Microsoft Windows dar. Die DLL enthält verschiedene Funktionen, die für das Erscheinungsbild der *Common Dialog Box* in bezug auf *Speichern*, *Speichern unter*, *Datei öffnen*, *Drucken* und dergleichen benötigt werden. Das Steuerelement garantiert ein immer gleiches Aussehen dieser Dialoge in jedem Windows Programm. Dabei ist die *Common*

*Dialog Box* letztlich auch wieder nur eine von Microsoft Windows bereitgestellte Schnittstelle zwischen Anwender und Software im Rahmen der COM-Technologie. Hier wird es für das Erscheinungsbild und den Aufruf der Hilfedatei zum Programm, *Dlthelp.hlp*, benötigt, was jedoch in Kapitel 1.5.1 auch mit Programmlisting 3 bereits erklärt wurde.

Die einzelnen Eingaben wurden, wie ebenfalls bereits erwähnt, in verschiedenen Gruppen eingeteilt. Zur besseren Übersicht wurde hierfür ein *TabStrip* Steuerelement verwendet, wie auf dem Formular zu sehen ist. Ein Register-Steuerelement (*TabStrip*) dient einem ähnlichen Zweck wie die Trennseiten eines Notizbuchs. Mit einem Register-Steuerelement können in einer Anwendung mehrere Seiten für denselben Fenster- oder Dialogfeldbereich definiert werden. Auf dem Registerblatt Steuerelement sind fünf Bildfelder erkennbar, die in einem Steuerelementfeld erstellt wurden.

- Allgemeine Systemangaben und Systemabschnitte
- Systempunkte
- Abschnittslasten
- Punktuelle Lasten
- Lagerabsenkungen und -verdrehungen

In jedem Bildfeld wurden die jeweiligen Gruppen von Steuerelementen, wie Textfelder Labelfelder oder FlexGrid[4] Steuerelemente, für die Eingaben des Benutzers zusammengefasst. Je nach Tabellenblatt, das angeklickt wurde, kann somit das zugehörige Bildfeld eingeblendet werden.

Steuerelementfelder sind Gruppen von Steuerelementen, deren Name identisch ist, die sich jedoch in einer Index-Nummer in Klammern hinter dem Namen des Steuerelements unterscheiden. Sie haben einen gemeinsamen Typ und verwenden gemeinsame Ereignisprozeduren. Jedes Steuerelement in einem Feld hat eine eindeutige Index-Nummer, mit der festgelegt werden kann, welches Steuerelement aus dem Feld auf ein Ereignis im Programm reagieren soll.

Ein Vorteil von Steuerelementfeldern liegt bei Ihrer Erstellung. Ein vorhandenes Steuerelement kann markiert und sehr schnell durch kopieren vervielfältigt werden. Die erzeugten Instanzen des Objekts erhalten dann eine fortlaufende Index-Nummer. Ebenso ermöglicht dies die leichte Abarbeitung von Code, wenn auf ein Ereignis reagiert werden soll. Für das stellte TabStrip entspricht die Indexnummer eines Tabellenblatts der Indexnummer eines Bildfelds, in dem die zugehörigen Steuerelemente für die Eingabe gruppiert sind.

Wie in den vorangehenden Kapiteln bereits erläutert wurde, wurde eine Eingabeoberfläche benötigt, die an Excel gebunden ist. Durch die Programmierung einer Dll wird zwar erreicht, dass die Anwendung nach dem Start im Thread von Excel läuft und beim Schließen von Excel auch die Dll terminiert wird, jedoch stellt die in den Speicher geladene Oberfläche selbst eine Form oder einen Rahmen dar, der vom Betriebssystem nicht automatisch in der Anwendung MS-Excel als Container angezeigt wird.

[4] Flexgrid Steuerelemente sind ebenfalls Steuerelemente in Visual Basic, die Eigenschaften für das Anzeigen und Bearbeiten von Werten in Tabellenform bieten.

Die bedeutet, dass die Eingabeoberfläche aus dem Blickfeld des Benutzers verschwindet, sobald vom Eingabeformular auf die Anwendung von Excel gewechselt wird. Will man dann wieder auf die Eingabeoberfläche zurückkehren, muss Excel minimiert werden, damit das Formular wieder sichtbar wird, wobei dann jedoch Excel nicht mehr sichtbar ist. Das gleiche Problem ergibt sich, wenn von Excel auf eine andere Anwendung gewechselt wird und später wieder zu Excel zurückgekehrt wird. Die Eingabeoberfläche für Durchlaufträger bleibt dann hinter einem anderen Fenster unsichtbar.

Damit der Anwender benutzerfreundlich mit der Software umgehen kann, muss also das geladene Formular nach dem Laden zu einem Child-Formular von Excel gemacht werden. Das wird mit Programmlisting 11 erreicht.

Die Anwendung MS-Excel stellt mit einem *MDI-Formular*[5] den Container für die Formulare oder Arbeitsmappen zur Verfügung. Jede geöffnete Arbeitsmappe in MS-Excel wird vom Betriebssystem ebenfalls als eigene Form behandelt. Damit die Child-Formulare nur innerhalb des MDI-Formulars gültig sind, muss die *MDI-Child*-Eigenschaft dieser Formen oder Arbeitsmappen auf „*true*“ gesetzt sein. Wird die Anwendung bzw. die Mutterform geschlossen, müssen auch alle geöffneten Arbeitsmappen geschlossen werden.

```
Zeile 1     Private Sub Form_Activate()          ‘ Beginn Form_Activate
Zeile 2     Dim hwndXL  As Long                  ' Zugriffsnr. für Windowsapp.
Zeile 3     Dim fenster As New cls_Window        ' Objektvar.
Zeile 4     Set wb = New cls_SheetHandle         ' Instanzieren der Klasse SheetHandle
Zeile 5     hwndXL = wb.DetectExcel              ' Ermitteln der Zugriffsnr. von XL
Zeile 6     fenster.mutterFenster hwndXL      ' macht frm_system zu Kindformular von XL
Zeile 7     Me.ZOrder 0                          ' Bringt Fenster in den Vordergrund
Zeile 8     fenster.MakeTopMost Me.hwnd          ' Zeigt frm_system immer im Vordergrund
Zeile 9     txt_Feld0.SetFocus                   ‘ setzt Focus auf txt_Feld
Zeile 10            End Sub                      ‘ Ende Form_Activate
```

Programmlisting 11.

Für Child-Formulare ergeben sich weitere Vorteile, da viele Aktionen bei Child-Formularen automatisch geregelt werden. Dies ist beispielsweise beim Schließen der Anwendung, beim Wechseln zu anderen Anwendungen oder wenn ein Fenster in den Hintergrund gebracht werden soll, der Fall.

Eine andere Möglichkeit, um das Eingabeformular erscheinen zu lassen, wäre es *modal* anzuzeigen. Modal oder auch „gebunden“ angezeigte Fenster auf dem Bildschirm müssen geschlossen werden, damit in einer anderen geöffneten Anwendung weitergearbeitet werden kann. Ein typisches Beispiel sind die *Speichern* oder *Speichern unter* Dialogfenster des Betriebssystem Microsoft Windows. Hierbei gilt zu beachten, dass modal angezeigte Formulare genau genommen nicht Inhalt des Containers sind, den ein MDI-Formular für ihre *Childs* darstellt. Für derart angezeigte Fenster entsteht dadurch mehr Verwaltungsaufwand,

[5] MDI – Multiple Document Interface, auch Mutterform genannt

wie z.B. beim Öffnen oder Schließen der Anwendung, da dies per Hand programmiert werden muss. Daher wurde diese Idee verworfen. Bei der Programmierung eines Out-of-Process Servers käme dies jedoch auf den Entwickler zu.

Für das TabStrip in Bild 116 wurden insgesamt fünf Tabellenblätter eingerichtet. Je nachdem welches Tabellenblatt des *TabStrips* angeklickt wurde, wird der Wert über einen Index abgefragt und auch ausgewertet. Programmlisting 12 zeigt die Vorgehensweise bei der Auswertung des Click Ereignisses auf das Tabellenblatt Steuerelement.

```
Zeile 1     Private Sub tbs_System_Click()       'Beginn Private Sub tbs_System_Click
Zeile 2     Dim Xcontrol As New cls_Controlhandle ' Objekt der Klasse cls_Controlhandle erstellen
Zeile 3     Dim i As Integer                     'Zählvariable
Zeile 4     Dim j As Integer                     'Zählvariable
```

Mit Zeile 5 wird der Indexwert des angeklickten Tabellenblatts des TabStrips (*tbs_System*) in die Variable *i* eingelesen.

```
Zeile 5     i = tbs_System.SelectedItem.Index – 1 ' Tab-Index abfragen
```

Mit der For-Schleife in Zeile 6 bis Zeile 9 werden alle Bildfelder, auf denen die Eingabewerte gruppiert wurden in den Hintergrund gebracht und deaktiviert.

```
Zeile 6     For j = 0 To tbs_System.Tabs.Count – 1       ' Schleife über alle Bildobjekte
Zeile 7       pic_system(j).Visible = False              ' Nicht benötigte Bildobjekte
ausblenden
Zeile 8     pic_system(j).Enabled = False                ' Bildobjekte deaktivieren
Zeile 9     Next
' Anzeigen des richtigen Bildobjekts und formatierungen einstellen
Zeile 10 Xcontrol.zeigeLastgruppe pic_system(i), tbs_System, Frame(i), cmd_cancel, cmd_ok, cmd_help
Zeile 11 End Sub                                         ' Ende Private Sub
tbs_System_Click()
```

Programmlisting 12.

Mit Zeile 10 werden die für die Eingabe des Benutzers benötigten Steuerelemente an die Methode *zeigeLastgruppe* des erzeugten Objekts der Klasse *cls_Controlhandle* übergeben. Die Methode erledigt dann das Einrichten der Elemente auf der Oberfläche. In der Klasse *cls_Controlhandle* sind die Methoden gesammelt, welche für das Handling der Steuerelemente auf dem Formular verantwortlich sind. Dies wird im entsprechenden Kapitel zur Klasse noch erläutert.

*Wie durch die gleichen Indexnummern der Tabellenblätter des Registerblatt Steuerelements und Bildfeld Steuerelements zu erkennen ist, ermöglicht dies die leichte Abarbeitung von Code, wenn auf ein Ereignis reagiert werden soll.*

Der Code, der in der Eingabeform geschrieben wurde, beschränkt sich im wesentlichen auf Code, der für die Verwaltung und logischen Verknüpfungen der Steuerelemente auf dem Formular. Er enthält bis auf seinen Umfang keine Ungewöhnlichkeiten. So kann an dieser Stelle auf weitere Erklärungen verzichtet werden und auf den Programmcode selbst, der ebenfalls dokumentiert ist, verwiesen werden. Die nachfolgenden zwei allgemeinen Methoden wurden dennoch in der Eingabeform selbst programmiert.

```
Public Function CheckZahl(AnsiTasteNeu As Integer) As Integer
' Die Methode überprüft die Eingaben des Benutzers auf Richtigkeit wenn Doublewerte eingegeben wurden

Public Function CheckZahlInt(AnsiTasteNeu As Integer) As Integer
' Die Methode überprüft die Eingaben des Benutzers auf Richtigkeit wenn Integerwerte eingegeben wurden
```

Alle weiteren Prozeduren wurden in allgemeinen Modulen oder in Klassenmodulen programmiert. Dies erhöht zum einen die Übersichtlichkeit im Programmcode, da die Prozeduren über den Modulnamen leicht sortiert werden können. Zum anderen können in Prozeduren benötigte Variablen im Deklarationsteil eines herkömmlichen Moduls als *Public* deklariert werden und sind dadurch von jeder Stelle aus im Programm verfügbar.

### 3.2.3.3 Allgemeine Programmmodule

Allgemeine Module dienen hauptsächlich als Container für Deklarationen und Prozeduren die im gesamten Programm gültig sind, das heißt, von anderen Programmteilen gemeinsam genutzt werden. Sie haben aber auch einen großen Einfluss auf die Übersichtlichkeit des Programmcodes, da verschiedene Prozeduren in unterschiedlichen Programmteilen, die ähnliche Funktionen erfüllen, unter einem aussagekräftigen Namen zusammengefasst werden können.

Die Komponente *edtra_DltVB.dll* umfasst lediglich ein allgemeines Module, *mod_dlt*. Hierin sind lediglich Variablen- und Funktionsdeklarationen untergebracht. Weiterhin hat das Modul keine einzige Methode, da ansonsten der Code in den unten erläuterten Klassenmodulen programmiert wurde. Dies wurde so gehandhabt, da in Visual Basic Beschränkungen für die Deklarationen von Public-Objektvariablen auf Formularebene bestehen und anfangs nicht klar war ob die Eingabeoberfläche aus einem oder mehreren Formularen bestehen wird.

```
Zeile 1   Option Explicit
Zeile 2   Public boolPunkt As Boolean       ' für Verhalten von Comandbutton auf pic_system(1)
Zeile 3   Public boolAbschnitt As Boolean       ‘ für Verhalten von Commandbutton auf pic_system(2)
Zeile 4  Public globalControl As cls_Controlhandle       ‘ Objektvar. für Klasse
cls_Controlhandle
Zeile 5  Public berechnen As cls_berechnen       ‘ Objektvar. für Klasse cls_berechnen
Zeile 6  Public worksheet As cls_SheetHandle       ‘ Objektvar. für Klasse cls_Sheethandle
Zeile 7  Public wb As cls_SheetHandle       ' Var. für aktives Workbook
Zeile 8  Public hoehe As Long       ' zum Einrichten des Formulars auf dem
```

```
Zeile 9  Public weite As Long                          ' Bildschirm bei Resizeereignissen
' Deklarieren der nötigen API-Routinen zum Steuern von Excel
Zeile 10 Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
                    (ByVal lpClassName As String, ByVal lpWindowName As Long) As Long
'Sendet Nachricht an ein Fenster,umgeht Schleife im Betriebssystem
Zeile 11 Public Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
                    (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long _
                    , ByVal lParam As Long) As Long
' Funktion wechselt ein Elternfenster eines Child-Fensters oder macht ein MDI-Formular zu einem
Mutterfenster
‘ eines Fensters
Zeile 12 Public Declare Function SetParent Lib "user32" (ByVal hWndChild As Long, _
                    ByVal hWndNewParent As Long) As Long
'Funktion positioniert ein geladenes (Fenster mit Atributen:Topmost,normal)
Zeile 13 Public Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, _
                    ByVal hWndInsertAfter As Long, ByVal X As Long, Y, ByVal cx As Long, _
                    ByVal cy As Long, ByVal wFlags As Long) As Long
```

Programmlisting 13.

In Zeile 1 ist die *Option Explicit* Anweisung zu sehen. Mit ihr wird auf Modulebene die explizite Deklaration aller Variablen in diesem Modul erzwungen. Wird die Anweisung nicht verwendet, deklariert VB die verwendeten und nicht deklarierten Variablen als Typ *Variant*, was alles bedeuten kann. Der Datentyp verhält sich sozusagen wie ein Chamäleon, er kann für viele verschiedene Datentypen in unterschiedlichen Situationen stehen. VB übernimmt auch dann die Umwandlung, wenn die Variablen an andere Prozeduren übergeben werden. Dies ist jedoch nicht unbedingt sinnvoll, da eventuell unnötig Speicherplatz belegt wird, die Übersichtlichkeit darunter leidet und Visual Basic die Daten mit einer expliziten Deklaration effizienter handhaben kann.

Mit Zeile 2 und Zeile 3 werden öffentliche boolesche Variablen deklariert, die je nach Zustand der Textboxen auf dem Formular *frm_system* auf *true* oder *fals* gesetzt sind. Wenn bereits Werte in den Boxen stehen, sind die Variablen auf true gesetzt und es wird verhindert, dass diese überschrieben werden, wenn der Anwender beispielsweise einen bereits eingegebenen Abschnitt bearbeiten möchte und hierzu die Werte aus dem Anzeigebereich in die Textboxen übernehmen will.

Zeile 4 bis Zeile 7 zeigt die Deklaration von Objektvariablen für unterschiedliche Klassen, wobei hier wiederum nur wieder Speicherplatz reserviert wird. Die eigentliche Instanzierung der Objekte erfolgt im Code des Formulars *frm_system*, wenn die Objekte tatsächlich benötigt werden. Dies ist beispielsweise bei der Einrichtung der Bildboxen auf dem Formular oder der Ausgabe der berechneten Ergebnisse im Excel Arbeitsblatt der Fall.

Die Variablen *hoehe* und *weite* in den nächsten beiden Zeilen werden benötigt um beim *Load Ereignis* des Formulars die aktuelle Höhe und Breite des Formulars einzulesen. Bei einer

Änderung der Formulargröße, beispielsweise beim Minimieren oder Maximieren des Formulars, als Aktion des Benutzers wird das *Resize* Ereignis des Formulars ausgelöst und die Form kann über die beiden Variablen wieder in der ursprünglichen Größe angezeigt werden. Man könnte hier auch den Pixelwert direkt eingeben, jedoch müsste dann bei einer späteren Änderung des Formulars bei Erweiterungen und dergleichen überall im Code die Zahlwerte ändern, was sehr umständlich ist.

Die restlichen Zeilen des Code sind Deklarationen von API-Funktionen von denen bereits die Rede war. Um ihre Funktionalität nutzen zu können, muss in der obigen Weise im Programm mit einer *Declare* Anweisung ein Verweise erstellt werden. Um DLL's, bzw deren Funktionen aus der *WIN32 API* verwenden zu können, müssen verschiedene Dinge beachtet werden. Sie können mit *Declare* im Deklarationsabschnitt eines Formulars oder Moduls deklariert werden, damit Visual Basic weiß, in welcher DLL die Funktion zu finden ist und womit dann auch festgelegt ist, welche Argumente übergeben werden müssen bzw. wie der Rückgabewert, wenn vorhanden, übergeben wird. Das angeschlossene Schlüsselwort *Function* wird wie gewöhnlich in VB für Funktionen verwendet, die einen Rückgabewert besitzen. *Sub* würde man verwenden, wenn die aufgerufene Prozedur keinen Rückgabewert liefert. Mit *Lib* (engl.: Library) und *Kernel32* in Anführungszeichen ist der Name der DLL bzw. der Bibliothek angefügt.

*user32.dll* ist eine Programmbibliothek, in der Prozeduren zur Verwaltung der Benutzeroberfläche, wie Ereignisse für Maus, für Fenster oder Menüs vorliegen. Eine Liste der wichtigsten Bibliotheken befindet sich im Anhang E der Arbeit.

Falls, wie es in Zeile 10 aus Programmlisting 13 der Fall ist, der genaue Name der Funktion vom deklarierten Namen abweicht, wird das Schlüsselwort *Alias,* gefolgt vom exakten Namen der Funktion verwendet. Dieser steht wieder in Anführungszeichen. Die Funktion hätte also auch gleich mit dem exakten Namen angesprochen werden können, jedoch existieren zum einen mehrere Varianten der Funktion in der Bibliothek. Dies kommt durch den Anhang *A* zum Ausdruck und liefert als Rückgabewert unterschiedliche Formate. Zum anderen wurde die Funktion direkt über den API Viewer (Bild 117) von VB eingefügt und die von VB gelieferte Syntax beibehalten.

Die Definition der zu übergebenden Argumente und des Rückgabewerts erfolgt dann wieder nach den gleichen Regelungen wie in VB üblich. Die Prozedur *FindWindow* kann beispielsweise feststellen, ob derzeit eine Anwendung auf Ihrem System ausgeführt wird. Sie akzeptiert zwei Zeichenfolgenargumente, eines für den Klassennamen der Anwendung, das andere für die Beschriftung der Titelleiste des Fensters.

*Die Verwendung der Funktion kann dann auf folgende Weise im Programmcode an anderer Stelle erfolgen:*

```
Dim hwnd As Long                    ' Deklaration einer Long Variable für den Rückgabewert
hwnd = FindWindow("XLMAIN", 0)
```

Der Rückgabewert ist eine Variable vom Typ *Long* der die Zugriffsnummer auf das Fenster oder die Anwendung enthält.

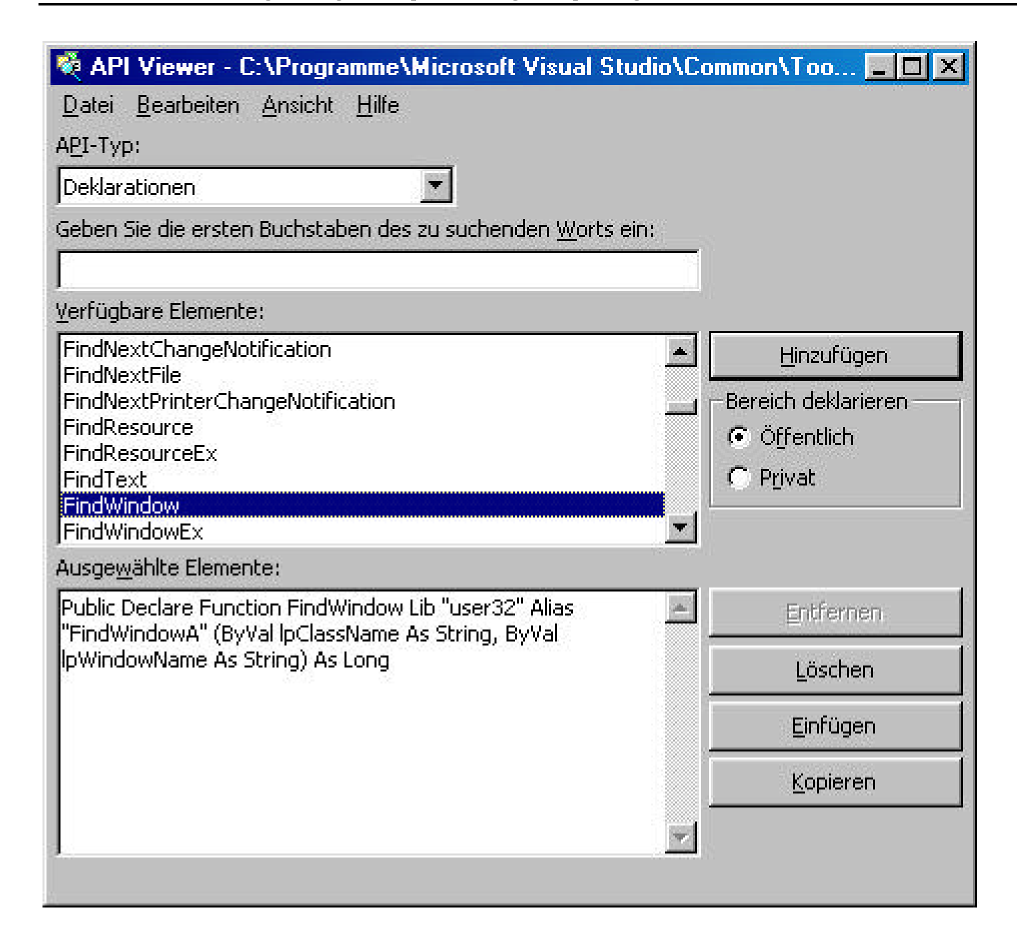



Bild 117. Der API Viewer der Entwicklungsumgebung von Visual Basic

### 3.2.3.4 Klassenmodule

#### 3.2.3.4.1 Allgemeines

Klassenmodule bilden die Grundlage der objektorientierten Programmierung in VB. Allerdings gibt es bei Visual Basic Einschränkungen gegenüber einer vollständig objektorientierten Programmiersprache. Dennoch bilden auch hier die Klassenmodule die Basis für jedes Objekt, auf die sich die Instanzen einer Klasse beziehen. Man könnte sie auch als einen Container oder Sammelplatz von Deklarationen, Eigenschaften, Methoden und Ereignissen bezeichnen, welche ein Objekt „beschreiben“. Innerhalb der das Objekt beschreibenden Klasse sind die Daten und Prozeduren gekapselt. In der Bibliothek *edtra_DltVB.dll* wurden insgesamt sechs Klassen programmiert, in der Methoden für unterschiedliche Funktionalität gesammelt sind. Die Klassen wurden programmintern zur besseren Identifikation mit dem Präfix *cls* bezeichnet.

#### 3.2.3.4.2 Klassenmodul cls_testDlt

Die Klasse bildet die Schnittstelle der Dll nach aussen und enthält mit *Test_Dlt()* wie oben beschrieben lediglich eine einzige Methode. Über sie wird das Eingabefenster initialisiert und in den Speicher geladen.

```
Option Explicit                    ' Option Explicit Anweisung
Dim fenster As frm_System          ‘ Deklaration einer Objektvar. für das Eingabeformular
Public Sub Test_dlt()              ‘ Beginn Methode Test_dlt
   Set fenster = frm_System        ‘ Zuweisung des
   fenster.Show                    ‘ Anzeigen des Formulars
End Sub                            ‘ Ende Test_dlt
```

Programmlisting 14.

Die Objektvarialble *fenster* ist hier lediglich klassenintern gültig deklariert. Dies reicht aus, da die Aggregation der Objekte der anderen Klassen im Formular selbst stattfindet wie das Klassendiagramm von Bild 113 zeigt. Das Formular bildet somit den eigentlichen Kern der Anwendung.

### 3.2.3.4.3 Klassenmodul *cls_Window*

Die Klasse enthält die Funktionalität, welche die Steuerung der Formulars in seinem globalen Verhalten betrifft.

**Attribute**

```
Private Const HWND_TOPMOST = -1          ' Plaziert Fenster ganz oben, auch wenn deaktiv
Private Const HWND_NOTOPMOST = -2        ' Plaziert das Fenster über allen non-topmost
                                         ' Fenstern(hinter den Topmost Fenstern.
Private Const SWP_NOSIZE = &H1                 ' Behält die aktuelle Größe bei
Private Const SWP_NOMOVE = &H2           ' Beeibehalten der momentanen Position (ignorieren der
X-
                                         ’ und Y-Parameter
Private Const SWP_NOACTIVATE = &H10            ’ Ignoriert die Aktivierung des Fensters
Private Const SWP_SHOWWINDOW = &H40            ’ Konstante zum Anzeigen des Fensters
Private Const TOPMOST_FLAGS = SWP_NOMOVE Or SWP_NOSIZE Or SWP_NOACTIVATE
Private Const HWND_BOTTOM = 1                  ’ Plaziert das Fenster ganz unten in der
Reihenfolge
Private Const HWND_TOP = 0               ’ Plaziert das Fenster ganz oben in der Reihenfolge
```

**Methoden**

Setzt die Reihenfolge des Fenster wieder auf den ursprünglichen Zustand zurück:

```
Public Sub MakeNormal(lngHwnd As Long)
        SetWindowPos lngHwnd, HWND_NOTOPMOST, 0, 0, 0, 0, TOPMOST_FLAGS
End Sub
```

Folgende Methode platziert das Fenster im Vordergrund und behält diese Position bei

```
Public Sub MakeTopMost(lngHwnd As Long)
        SetWindowPos lngHwnd, HWND_TOPMOST, 0, 0, 0, 0, TOPMOST_FLAGS
End Sub
```

Die Methode positioniert das Fenster unter allen anderen Fenstern:

```
Public Sub MakeBottom(lngHwnd As Long)
        SetWindowPos lngHwnd, HWND_BOTTOM, 0, 0, 0, 0, TOPMOST_FLAGS
End Sub
```

'Die Methode macht aus dem geladenen Fenster ein Unterfenster der Excel Anwendung:

```
Public Sub mutterFenster(hwndNew As Long)
  Dim X As Long                                  'Rückgabewert der API
        SetParent frm_System.hwnd, hwndNew
End Sub
```

Programmlisting 15.

### 3.2.3.4.4 Klassenmodul cls_Controlhandle

In dieser Klasse sind die Methoden für die Steuerung der Steuerelemente auf dem Formular *frm_System* gesammelt.

**Attribute**

```
Dim neuerAbschnitt()              ' Integer Array zum Auslesen von Feld-/Abschnittsnummer u. Reihe in
die
                                  ' geschrieben werden soll, wenn aus cmd_newAbschnitt aufgerufen
Dim boolNeuerAbschnitt As Boolean  ' Dient für die Anzeige,dass ein Abschnitt eingefügt werden soll
Dim Feld_P()                      ' Array für Punktlasten, nur inerhalb Klasse gültig
Dim Feld_S()                      ' Array für Linienlasten, nur inerhalb Klasse gültig
```

**Eigenschaften**

In der Klasse wurden Eigenschaften über Property-Anweisungen[6] deklariert, über die Werte in einer Klasse gelesen oder verändert werden können. Dies wurde erforderlich, da in Visual Basic nicht direkt auf Arrays zugegriffen werden kann. Die *Property Get* Eigenschaft dient zum Lesen der Eigenschaft, über die *Property Let* Eigenschaft kann eine Eigenschaft geändert werden.

Die *Property Get* Anweisung dient zum Lesen der Eigenschaft:

```
Public Property Get boolnewabschnitt() As Boolean
```

[6] Property-Anweisung – Über diese Anweisungen können Eigenschaften mit bestimmten Attributen in einem Klassenmodul deklariert werden

```
        boolnewabschnitt = boolNeuerAbschnitt
End Property
```

*Property Let* Dient zum Verändern der Eigenschaft:

```
Public Property Let boolnewabschnitt(bool As Boolean)
        boolNeuerAbschnitt = bool
End Property
```

*BoolNeuerAbschnitt* ist eine Variable vom Typ Boolean, die angibt, dass ein neuer Abschnitt mit dem Popupmenü eingefügt werden soll. Der boolesche Ausdruck wird in verschiedenen Routinen benötigt, um die richtigen Werte einzutragen.

Diese Methode dient zum Lesen der Eigenschaft:

```
Public Property Get einfügenAbschnitt() As Integer()
        einfügenAbschnitt = neuerAbschnitt()
End Property
```

Diese Methode dient zum Verändern der Eigenschaft:

```
Public Property Let einfügenAbschnitt(feld() As Integer)
        neuerAbschnitt = feld()
End Property
```

Übergeben wird Feld mit Feld-Nr. / Anzahl der Unterabschnitte / Anzahl Abschnitte. *neuerAbschnitt()* ist ein Array aus Integerwerten und dient zum Auslesen von Feld- und Abschnittsnummer und der Reihe in der Tabelle, in welche die Werte geschrieben werden sollen. Der Aufruf erfolgt aus dem Klick-Ereignis auf den Command Button[7] *cmd_newAbschnitt.*

**Methoden**

Die Klasse cls_Controlhandle enthält 17 weitere Methoden, die im Folgenden kurz aufgeführt sind. Die Erklärung der einzelnen Funktionen findet immer unterhalb des Methodenkopfes statt.

```
Public Function Wert(Text As String) As String
```

In der Methode werden die Eingaben formatiert, bevor sie in die Tabellen geschrieben werden. Es findet eine Überprüfung auf Punkt und Komma statt. Nullen am Ende werden hinzugefügt, wenn nötig, und am Anfang abgeschnitten. Mit dem Funktionsaufruf wird ein Array mit den Werten übergeben, die reihenweise in die Tabelle eingetragen werden. Der Rückgabewert der Methode ist wiederum ein Array mit den formatierten Eingaben.

```
Public Function schreibe_flex(flgNew As MSFlexGrid, indexNew As Integer, reiheNew As Integer, _
feldNew() As String)
```

[7] Command Button – Befehlsschaltfläche. Steuerelement in Visual Basic

Die Methode *schreibe_flex()* ist zuständig um Eingabewerte in eine Tabelle zu schreiben. An die Methode werden als Parameter das Objekt eines Tabellensteuerelements, der Index der Tabelle, die Reihe, in welche die Werte geschrieben werden und ein Array aus Stringwerten übergeben. Rückgabewerte existieren nicht.

```
Public Sub getLast(flgNew As MSFlexGrid, indexNew As Integer, reiheNew As Integer, formNew As
Object)
```

*getLast()* liest die Werte aus einer Tabelle aus und schreibt sie in die Textboxen zur Bearbeitung. Als Parameter übergeben werden ein Verweis auf ein Tabellensteuerelement, die Indexnummer der Tabelle, die Reihe, aus der die Werte ausgelesen werden, und ein Verweis auf das Objekt der Eingabeoberfläche.

```
Public Function sortieren(flgNew As Control, reiheNew) As Control
```

Die Methode wird beispielsweise benötigt, wenn Abschnitte aus einem Träger entfernt wurden. Liegt der entfernte Abschnitt irgendwo in Trägermitte müssen die nachfolgenden Abschnitte oder Felder neu sortiert und nummeriert werden. Übergeben wird ein *FlexGrid* Steuerelement und die Reihennummer, ab der das Grid neu sortiert werden muss. Rückgabewert ist das neu Sortierte *FlexGrid* Objekt.

```
Public Sub zeigeLastgruppe(picNew As PictureBox, tbsNew As TabStrip, fraNew As Frame, _
cancelNew As CommandButton, okNew As CommandButton, helpNew As CommandButton)
```

Die Methode dient zum Einrichten und Anzeigen der Bildfelder mit den Tabellensteuerelementen für die jeweiligen Eingabegruppen. Übergabeparameter sind verschiedene Steuerelemente auf der Eingabeoberfläche, die je nach angewähltem Eingabebereich in den Vordergrund oder Hintergrund gebracht werden.

```
Public Function leereFlex(flgNew As MSFlexGrid, reiheNew)
```

Die Methode setzt die Einträge in den Tabellensteuerelementen zurück, wenn Werte geändert und gelöscht werden. Übergabeparameter ist ein Objektverweis auf die Tabelle, und die Reihe, in der die Werte zurückgesetzt werden sollen.

```
Public Sub leereTextbox(formNew As Object, Index As Integer)
```

Die Methode setzt die Einträge in den jeweiligen Textboxen der Eingabegruppen zurück, nachdem die Werte in die Tabellen geschrieben wurden. Übergabeparameter ist ein Objektverweis auf die Eingabeoberfläche und der Index der Eingabegruppe in der die Werte in den Textboxen zurückgesetzt werden sollen.

```
Public Function checkflex(frmNew As Object, indexNew As Integer) As String()
```

Die Methode ermittelt die Feldnummer und die Abschnittsnummer in der letzten Reihe der Tabelle für die Eingabe der Systemabschnitte. Zudem wird eine Indexnummer ermittelt, in welche die neuen Einträge geschrieben werden sollen. Übergabeparameter sind ein Verweis auf das Objekt der Eingabeoberfläche und ein Index für die Tabelle, die überprüft werden soll, der hier nicht unbedingt nötig gewesen wäre, jedoch für eine spätere eventuelle Erweiterung des Programms beibehalten wurde. Als Rückgabewert wird ein eindimensionales Array übergeben, in dessen ersten Feld die Feldnummer des letzten Abschnitts in der Tabelle steht. Im zweiten Feld steht die Abschnittsnummer und im dritten Feld die aktuelle Reihe, in welcher der neue Eintrag stattfinden soll.

Public Function checkflexOld(frmNew As Object, indexNew As Integer, reiheNew As Integer) As String()
Die Methode ermittelt die Feldnummer und die Abschnittsnummer in der Reihe der Tabelle für die Eingabe der Systemabschnitte, wenn Werte aus der Tabelle geändert werden sollen. Übergabeparameter sind ein Verweis auf das Objekt der Eingabeoberfläche und ein Index für die Tabelle, die überprüft werden soll. Als Rückgabewert wird ein eindimensionales Array übergeben, in dessen ersten Feld die Feldnummer des bearbeiteten Abschnitts in der Tabelle steht. Im zweiten Feld steht die Abschnittsnummer.
Public Function getTextbox(formNew As Object, Index As Integer) As String()

Die Methode liest die Werte aus den unterschiedlichen Eingabebereichen in ein Datenfeld für die weitere Bearbeitung ein. Als Parameter übergeben werden ein Objektverweis auf die Eingabeoberfläche und ein Index für den Bereich, aus dem die Werte gelesen werden sollen. Als Rückgabewert wird ein Array aus Stringwerten geliefert, der die Werte aus den Textboxen enthält und in der letzten Zelle des Array Angabe liefert, ob es sich um eine fehlerhafte Eingabe handelt oder nicht.

Public Function textbox_lostfocus(frmNew As Object, Index) As Integer
Die Methode prüft auf Werte in den jeweiligen Eingabebereichen auf Fehler, wenn der Focus auf ein Tabellensteuerelement übergeht. Dies wird erforderlich, da der Commandbutton, verschiedene Zustände für das Bearbeiten und Übernehmen von Werten besitzt. Stehen Werte in den Textboxen und der Fokus geht auf ein Tabellensteuerelement über, darf der Zustand für den Commandbutton nicht geändert werden.

Public Function selectAbschnittslast(formNew As Object, Index As Integer, feldNew() As String) As Integer()
Die Methode ermittelt die Reihe im Tabellensteuerelement, in welche die eingegebenen Werte geschrieben werden sollen, wenn eine neue punktuelle Last eingegeben wurde. Als Parameter werden übergeben, ein Objektverweis auf die Eingabeoberfläche, ein Index für den Bereich, aus dem die Werte gelesen werden sollen und ein Feld mit Strings der eingegebenen Werte. Als Rückgabewert wird ein Array aus Integerwerten geliefert, das im ersten Feld angibt, ob der Tabelle eine neue Zeile hinzugefügt werden muss und im zweiten Feld um welche Zeile es sich handelt.

Private Function prüfeLastfall(flex As MSFlexGrid, Index As Integer, feldNew() As String) As Integer()
Da die Abschnitts- und Punklasten jeweils nach Lastfall sortiert angezeigt werden, wird diese Methode benötigt. Als Rückgabewert wird ein Array aus Integerwerten geliefert, das im ersten Feld anzeigt, ob der eingegebene Lastfall existiert und im zweiten und dritten Feld den Wert für die unterste beziehungsweise oberste Reihe mit Indexnummer für den gleichen Lastfall enthält.

Private Function prüfeFeld(flex As MSFlexGrid, Index As Integer, reihe() As Integer, _
feldNew() As String) As Integer()
Da die Abschnitts- und Punktlasten auch nach der Feldnummer sortiert angezeigt werden, wird diese Methode benötigt. Sie wird bei fehlerfreien Werten direkt nach *prüfeLastfall()* aufgerufen. Als Rückgabewert wird ein Array aus Integerwerten geliefert, das im ersten Feld anzeigt, ob der eingegebene Lastfall existiert und im zweiten und dritten Feld den Wert für die unterste beziehungsweise oberste Reihe mit Indexnummer für die gleiche Feldnummer enthält.

Private Function prüfeAbschnitt(flex As MSFlexGrid, Index As Integer, reihe() As Integer, _
feldNew() As String) As Integer()

Da die Abschnitts- und Punktlasten zudem nach der Abschnittsnummer sortiert angezeigt werden, wird diese Methode benötigt. Sie wird bei fehlerfreien Werten direkt nach *prüfeFeld()* aufgerufen. Als Rückgabewert wird ein Array aus Integerwerten geliefert, das im ersten Feld anzeigt, ob eine neue Zeile hinzugefügt werden muss, im zweiten und dritten Feld den Wert für die unterste und oberste Reihe mit Indexnummer für die gleiche Abschnittsnummer enthält.

Public Function eintrag_löschen(formNew As Object, Index As Integer, reihe As Integer)

Die Methode wird aus dem Popupmenü aufgerufen und ist für das Löschen von Systemabschnitten zuständig. Übergabeparameter an die Methode sind ein Objektverweis auf die Eingabeoberfläche, ein Index, um welches Tabellensteuerelement es sich handelt, und eine Nummer für die Reihe, die gelöscht werden soll.

Public Function eintrag_einfügen(formNew As Object, Index As Integer, reihe As Integer)

Die Methode wird aus dem Popupmenü aufgerufen und ist für das Einfügen von Systemabschnitten zuständig. Übergabeparameter an die Methode sind ein Objektverweis auf die Eingabeoberfläche, ein Index, um welches Tabellensteuerelement es sich handelt, und eine Nummer für die Reihe, die gelöscht werden soll.

### 3.2.3.4.5 Klassenmodul *cls_berechnen*

In dieser Klasse findet das Auslesen der Ergebnisse aus der eigentlichen Berechnungskomponente statt. Um später aus anderen Klassen auf die Ergebniswerte zugreifen zu können, wurden verschiedene Eigenschaften deklariert, über die auf die Ergebnisse, die in mehrdimensionalen Arrays vorliegen, zugegriffen werden kann. Unten ist der Deklarationsteil der Klasse aufgelistet. Hier werden die Arrays deklariert und beschrieben, auf welche über die Property-Eigenschaften aus anderen Klassenmodulen zugegriffen werden kann.

**Attribute**

```
Option Explicit
Dim Dlt As DltV2                                ' Objektvariable für die Java-Berechnungs-dll
Dim impact() As Integer                   ' Array für Lastfälle
Dim solution() As Double                        ' Array für Ergebnisse
Dim überlagerung() As Double              ' Array für die Ergebnisse aus der Lastfallüberlagerung
Dim auflager() As Double                        ' Array für die Ergebnisse der Auflager Einzellastfälle
Dim auflagerÜberlagerung() As Double      ' Array für die Ergebnisse der Auflager Lastfallüberlagerung
Dim lastfallAnz() As Integer              ' Für Rückgabewerte Feldanzahl und Anzahl versch. Lastfälle
Dim maxAbschnitt() As Double              ' Anzahl der Abschnitte
Dim maxFeldmoment() As Double                   'Array für die maximalen Feldmomente
```

**Eigenschaften**

Nachfolgend sind die einzelnen Property Get Anweisungen aufgeführt, die Lesezugriff auf die Ergebniswerte erlauben. Eine Auflistung der Property Let Anweisungen erfolgt hier nicht, da die Anweisungen zum Verändern der Eigenschaften dienen und nicht verwendet werden.

```
Public Property Get feldAbschnitte() As Double()
' dient zum Lesen der Eigenschaft.Übergeben wird Feld mit Feld-Nr.|Anzahl Unterabschnitte|Anzahl Abschnitte
        feldAbschnitte = maxAbschnitt()
End Property
```

*maxabschnitt()* ist ein dreidimensionales Array aus Double Werten, welches in der ersten Dimension implizit über den Index die Feldnummer angibt. In der zweiten Dimension sind die Abschnitte mit ihrer Nummer eingetragen. In der dritten Dimension ist in Zelle 1 die Abschnittsnummer, in Zelle 2 die Anzahl der Unterabschnitte des Abschnitts und in der dritten Zelle der die Länge der Unterabschnitte eingetragen.

```
Public Property Get auflagerErgebniss() As Double()
'dient zum Lesen der Eigenschaft.Übergeben wird Feld mit den Ergebnissen der Aulagerkräfte
        auflagerErgebniss = auflager()
End Property
```

*auflager()* ist ein dreidimensionales Array aus Double Werten für die Auflagerkräfte der Einzellastfälle, welches in der ersten Dimension implizit über den Index die Lastfallnummer angibt. In der zweiten Dimension sind die Felder mit ihrer Nummer eingetragen. In der dritten Dimension ist in Zelle 1 die Feldnummer, in Zelle 2 der Wert für die Querkraft und in Zelle 3 der Wert für das Moment eingetragen.

```
Public Property Get auflagerErgebnissÜberlagerung() As Double()
'dient zum Lesen der Eigenschaft.Übergeben wird Feld mit den Ergebnissen der Aulagerkräfte
        auflagerErgebnissÜberlagerung = auflagerÜberlagerung()
End Property
```

*auflagerÜberlagerung()* steht für die Werte der Auflager aus der Lastfallüberlagerung, wobei das Array aus Double Werten lediglich zweidimensional ist. In der ersten Zelle der zweiten Dimension wird die Feldnummer eingetragen, danach die Werte für die maximale und minimale Querkraft, sowie das maximale und minimale Moment in genannter Reihenfolge.

```
Public Property Get Lastfall() As Integer()
'dient zum Lesen der Eigenschaft.Übergeben wird Feld mit sortierten Lastfällen
'impact(0) ist leer. Ubound ergibt die Anzahl der versch. Lastfälle
        Lastfall = impact()
End Property
```

*impact()* ist ein eindimensionales Array aus Integer Werten. Die erste Zelle enthält keine Werte. Die Zellen 1 bis zur oberen Grenze des Array enthalten die Lastfallnummern.

```
Public Property Get ergebnisse() As Double()
'dient zum Lesen der Eigenschaft.Übergeben wird Feld mit sortierten Lastfällen
'impact(0) ist leer. Ubound ergibt die Anzahl der versch. Lastfälle
        ergebnisse = solution()
End Property
```

*solution()* ist wieder ein dreidimensionales Array aus Double Werten für die Ergebnisse der Schnittgrößen und Punktverformungen der Einzellastfälle. Die erste Dimension läuft über alle Lastfälle. Die zweite Dimension über alle Unterabschnitte. In der dritten Dimension sind in Zelle 1 die Feldnummer in Zelle 2 die Nummer des Unterabschnitts eingetragen. Hierbei ist zu beachten, dass für die Ergebnisse die Unterabschnitte in jedem Feld aufsummiert wurden. Die nachfolgenden Zellen enthalten die Werte für die Querkraft, das Moment und die Durchbiegung jeweils für den Abschnittsanfang und das Abschnittsende in angegebener Reihenfolge.

```
Public Property Get ergebnisseÜberlagerung() As Double()
'dient zum Lesen der Eigenschaft.Übergeben wird Feld mit sortierten Lastfällen
'impact(0) ist leer. Ubound ergibt die Anzahl der versch. Lastfälle
        ergebnisseÜberlagerung = überlagerung()
End Property
```

*überlagerung()* ist ebenfalls für die Werte der Schnittgrößen und Verformungen. Jedoch hier für die Lastfallüberlagerung. Das Array aus Double Werten ist aufgrund der fehlenden Lastfälle nur zweidimensional. In der ersten Dimension sind wieder die Unterabschnitte aufgetragen. In der ersten Zelle der zweiten Dimension ist die Nummer des Unterabschnitts eingetragen, in den nachfolgenden Zellen die Ergebniswerte für die minimale und maximale Querkraft, Moment und Durchbiegung in der angegebenen Reihenfolge.

```
Public Property Get minMax() As Double()
'dient zum Lesen der Eigenschaft.Übergeben wird Feld mit Feld-Nr.|Anzahl Unterabschnitte|Anzahl Abschnitte
        minMax = maxFeldmoment()
End Property
```

*maxFeldmoment()* ist ein zweidimensionales Array aus Double Werten für die Ergebniswerte der maximalen Feldmomente aus der Lastfallüberlagerung. In der erste Dimension ist die Feldnummer aufgetragen. In der zweiten Dimension in folgender Reihenfolge die Werte für die Feldnummer, das maximale Feldmoment, den X-Wert des Moments im gesamten Trägerverlauf und der X-Wert des Moments im betreffenden Feld.

### Methoden

Nachfolgend sind die sieben Methoden der Klasse und deren Aufgaben aufgeführt.

`ermittleLastfall(formNew As Object) As Integer()`

In der Methode wird die Anzahl der Felder und die Anzahl der angegebenen Lastfälle ermittelt. Als Rückgabewert wird ein Array aus Integer Werten geliefert. Zudem wird in ein weiteres, eindimensionales Array aus Integerwerten (*lastfallAnz()* ) die Nummer des Lastfalls eingetragen, wobei das erste Feld des Array leer bleibt und der Eintrag der Werte beim Index 1 beginnt. Diese Werte werden in der Methode *übergabe()* benötigt, um die Eingabewerte an die Berechnungsroutine zu übergeben.

Public Sub sortLastfall()
Die Methode wird aus *ermittleLastfall()* aufgerufen und sortiert die Lastfallnummern in *lastfallAnz()* in aufsteigender Reihenfolge.

```
Zeile 1  Private Function transformLager(lagerNew As String) As String    ‘ Beginn transformLager
'Umwandlung der Lagerbedingungen in Zahlenwerte (Strings)
Zeile 2  Select Case lagerNew                      ‘ Beginn Select Case Block
Zeile 3    Case "g"                                ‘ Wenn gelenkiges Lager eingegeben wurde
Zeile 4     lagerNew = 1
Zeile 5    Case "e"                                ‘ Wenn eingespanntes Lager eingegeben wurde
Zeile 6     lagerNew = 2
Zeile 7    Case "f"                                ‘ Wenn freies Lager eingegeben wurde
Zeile 8     lagerNew = 3
Zeile 9    Case "s"                                ‘ Wenn schwebend eingespanntes Lager eingegeben
wurde
Zeile 10    lagerNew = 4
Zeile 11 End Select                                ‘ Ende Select Case Block
Zeile 12 transformLager = lagerNew                 ‘ Rückgabewert übergeben
Zeile 13 End Function                              ‘ Ende transformLager
```

Programmlisting 16.

Die Methode wandelt die Strings aus den Comboboxen der Eingabeform nach dem Einlesen in Integerwerte um. Die zugewiesene Nummer kann oben entnommen werden.

Public Function getWerte(formNew As Object)
In der Methode werden die Arrays für die Ergebniswerte eingerichtet oder dimensioniert. Sowie die Werte aus der Berechnungskomponente ausgelesen. Die Arrays, in welche die Ergebnisse eingelesen werden, sind bei den Property-Anweisungen erklärt.

Private Function leseLager(frmNew As Object) As String()
Die Methode wird aus *übergabe()* aufgerufen und liest die Werte für die allgemeinen Systemangaben aus der Eingabeform in ein eindimensionales Array ein. Hierbei wird in Feld 1 die Lagerung am Trägeranfang, in Feld 2 die Lagerung am Trägerende und in Feld 3 der Eintrag, ob es sich um eine Berechnung nach Theorie II. Ordnung handelt eingetragen.

Private Function leseAbschnitt(frmNew As Object, Index As Integer) As String()
In der Methode werden die Werte aus den einzelnen Tabellen ausgelesen. Die Rückgabe erfolgt nach einer Fehlerkorrektur in einem eindimensionalen Array.

Public Function Übergabe(frmNew As Object) As Integer

In der Methode werden die Eingabewerte an das eigentliche Berechnungsmodul übergeben. Nachfolgend sind die einzelnen Methodenaufrufe der eigentlichen Berechnungskomponente aufgeführt. Dlt ist eine Objektvariable, die einen Verweis auf die Berechnungs-Dll enthält. Die Variable *Dlt* selbst wurde im Deklarationsteil der Klasse als *Private* deklariert (siehe oben) und ist damit im gesamten Klassenmodul verfügbar, jedoch nicht auserhalb. Die Initialisierung eines Objekts der Berechnungskomponente findet dann in der Methode *Übergabe()* statt. Der Code hierfür ist denkbar einfach:

```
Set Dlt = New DltV2                    ' Initialisierung eines Objekts der Berechnungskomponente
```

*DltV2* ist eine öffentliche Klasse der Berechnungskomponente *Dlt.dll*, und bietet Zugriff bei der Instanzierung der Komponente. Im folgenden sind die einzelnen Methodenaufrufe angezeigt, die nötig sind, um die eingegebenen Werte an die Berechnungseinheit weiterzugeben:

```
Dlt.initMatrix CInt(System(0)), CInt(System(1)), CInt(System(2)), lastfallAnz(1), lastfallAnz(0)
```

In *InitMatrix* werden die Werte für die allgemeinen Systemeingaben eingelesen.

```
Dlt.setSystemAbschnitt CInt(abschnitt(j, 0)), CInt(abschnitt(j, 1)), CDbl(abschnitt(j, 2)), _
CInt(abschnitt(j, 3)), CDbl(abschnitt(j, 4)) , CDbl(abschnitt(j, 5)), _
CDbl(abschnitt(j, 6)), CDbl(abschnitt(j, 7)), CDbl(abschnitt(j, 8)), _ddd
CDbl(abschnitt(j, 9))
```

*setSystemAbschnitt* ist der Aufruf für die Werte der Systemabschnitte.

```
Dlt.setSystemPunkt CInt(Punkt(i, 0)), CInt(Punkt(i, 1)), CInt(Punkt(i, 2)), _
CDbl(Punkt(i, 3)), CDbl(Punkt(i, 4)), CDbl(Punkt(i, 5)), CDbl(Punkt(i, 6))
```

*setSystemPunkt* ist der Aufruf für die Werte der Systemabschnitte.

```
Dlt.setLastenAbschnitt CInt(Slast(j, 0)), CInt(Slast(j, 1)), CInt(Slast(j, 2)), CDbl(Slast(j, 3)), _
CDbl(Slast(j, 4)), CDbl(Slast(j, 5)), CDbl(Slast(j, 6)), CDbl(Slast(j, 7))
```

*setLastenAbschnitt* ist der Aufruf für die Werte der Abschnittslasten.

```
Dlt.setLastenPunkt CInt(Plast(j, 0)), CInt(Plast(j, 1)), CInt(Plast(j, 2)), CInt(Plast(j, 3)), CDbl(Plast(j, 4)) _
, CDbl(Plast(j, 5)), CDbl(Plast(j, 6)), CDbl(Plast(j, 7))
```

*setLastenPunkt* ist der Aufruf für die Werte der Abschnittslasten.

```
Dlt.setLagerabsenkungVerdrehung CInt(senkung(i, 0)), CInt(senkung(i, 1)), _
CInt(senkung(i, 2)), CDbl(senkung(i, 3)), CDbl(senkung(i, 4))
```

*setLagerabsenkungVerdrehung* ist der Aufruf für die Übergabe der Werte der Lager-absenkungen und -verdrehungen.

```
Dlt.init
Dlt.berechnen
```

Mit *Init* und *berechnen* werden Konstruktoren aufgerufen, welche die Eingabewerte für die Berechnung aufarbeiten und die Berechnung gestartet wird.

### 3.2.3.4.6 Klassenmodul cls_SheetHandle

Die Klasse *cls_Sheethandle* steuert die Ausgabe der Ergebnisse in der Office-Anwendung. Attribute und Eigenschaften enthält die Klasse nicht. Die vier Methoden der Klasse sind nachfolgend erläutert.

```
Public Function fm(ByVal Zahl As Double, ByVal GesSt As Integer, ByVal DezSt As Integer) As String
```

Die Methode wird benötigt, um die Ergebniswerte für die Ausgabe in Excel zu formatieren. Übergeben wird die zu formatierende Zahl als Doublewert, die Gesamtstellenanzahl und die Dezimalstellenanzahl. Der Rückgabewert besteht aus einem String.

In *DetectExcel()* wird ein gültiger Zugriffscode auf die Office-Komponente ermittelt, wenn diese geladen ist und im Arbeitsspeicher vorliegt. Der Rückgabeparameter ist die Zugriffsnummer der Office-Komponente. Liegt „MS-Excel“ nicht im Arbeitsspeicher vor, liefert die Methode eine „0“ als Rückgabewert.

```
Zeile 1  Function DetectExcel() As Long                 ‘ Beginn DetectExcel
'Erkennt, ob Excel ausgeführt wird, und registriert dies.
Zeile 2  Const WM_USER = 1024                                   ' ID der Nachricht,die versandt werden
soll
Zeile 3  Dim hwnd As Long                               'Zugriffsnummer für die Excel Anwendung
' Wenn Excel läuft, wird eine Zugriffsnummer zurückgegeben.
Zeile 4  hwnd = FindWindow("XLMAIN", 0)
Zeile 5  If hwnd = 0 Then                                       ' 0 bedeutet, daß Excel nicht ausgeführt
wird
Zeile 6          Exit Function                          ’ Wenn Excel nicht läuft -> Funktion beenden
Zeile 7  Else
' Wenn Excel läuft. Verwenden der API-Funktion SendMessage, um Excel in der
’Tabelle ausgeführter Objekte einzutragen.
Zeile 8     SendMessage hwnd, WM_USER + 18, 0, 0        ‘ SendMessage umgeht Warteschleife im
Zeile 9  End If                                         ' Betriebssystem
Zeile 10 DetectExcel = hwnd                             ’ Rückgabe der Zugriffsnummer
Zeile 11 End Function                                   ’ Ende DetectExcel
```

Programmlisting 17.

In der Methode *GetExcel* werden die Ergebnisse schließlich in das aktuelle Tabellenblatt von Excel geschrieben. Hierbei wird das Excel Objekt wieder über die COM Technologie gesteuert. Da der Code einige hundert Zeilen enthält wird lediglich ein Ausschnitt wiedergegeben.

```
Zeile 1  Public Function GetExcel(formNew As Object, berechnen As Object)      ’ Beginn GetExcel
```

```
' Prüfen auf Microsoft Excel. Wenn Microsoft Excel ausgeführt wird, wird dies in die Tabelle
' ausgeführter Objekte eingetragen.
Zeile 2  hwnd = DetectExcel                         'ermittelt Zugriffsnummer von Excel
' Überprüfen, ob eine Kopie von Microsoft Excel bereits ausgeführt wird.
'On Error Resume Next                       'die  Fehlerbehandlung zurückstellen
' GetObject-Funktionsaufruf ohne erstes Argument gibt einen Verweis auf eine Instanz der Anwendung
, zurück. Wenn die Anwendung nicht ausgeführt wird, tritt ein Fehler auf.
Zeile 3  Set XL = GetObject(, "Excel.Application")            ' Var. XL initialisieren
Zeile 4  If Err.Number < > 0 Then boolshit = True             'boolsche Variable für Abbruch
Zeile 5  Err.Clear                                            'Err-Objekt im Fehlerfall löschen
Microsoft Excel mit zugehöriger Application-Eigenschaft einblenden. Fenster mit der Datei unter
Verwendung der Windows-Auflistung des XL-Objektverweises anzeigen
Zeile 6  XL.Application.Visible = True              ' Anzeigen der Oberfläche von Excel
Zeile 7  XL.Parent.Windows(1).Visible = True
' Wenn diese Kopie von Microsoft Excel beim Starten nicht ausgeführt wurde wird die Variable XL wieder
auf
' Nothing gesetzt.
Zeile 8  If boolshit = True Then
Zeile 9          Set XL = Nothing
Zeile 10         boolshit = False
Zeile 11 End If
```

Folgender Code steuert die Ausgabe der Ergebnisse auf dem aktiven Tabellenblatt in Excel. Der Zugriff erfolgt über die Variable XL. Innerhalb des *With*-Block wird dann über eigene Objekte, die Excel bereitstellt die Ausgabe gesteuert und das Tabellenblatt formatiert. Die Innerhalb des *With*-Blocks angesprochenen Objekte, Methoden , und Eigenschaften müssen wieder mit einem vorangestellten Punkt benannt werden. Der VBA Code der innerhalb der *With*-Blöcke steht, wurde in Excel selbst mit einem Makro aufgezeichnet und über die Zwischenablage in den Code der Anwendung kopiert. Dadurch, dass beim Aufzeichnen von Makros in einer Office Anwendung, jeder Schritt aufgezeichnet wird, enthält der Code dann zwar viele Zeilen, die unnötig sind und entfernt werden müssen. Jedoch erspart diese Vorgehensweise das mühselige Suchen der entsprechenden Objekte, Eigenschaften und Methoden aus dem Objektkatalog, aus Büchern oder aus der Online Hilfe zu VBA. Aber auch hierbei blieb das Suchen und die Einarbeitung nicht gänzlich erspart, da nicht alle der in den Makros aufgezeichneten VBA Konstanten von Visual Basic erkannt wurden.

```
Zeile 12 With XL
Zeile 13         With .Range(CStr(alphabet(0)) + CStr(startzeile))
```

```
Zeile 14.        Value = "Durchlaufträger"
     .
     .
     .
```

Programmlisting 18.

Die Methode *transformLager()* wandelt die eingegebene Lagerbedingung in den Comboboxen der Eingabeoberfläche in einzelne Buchstaben für die Ausgabe in Excel um. Die Methode ist identisch mit der Funktion, die in der Klasse cls_berechnen existiert und wurde aufgrund der Kapselung und der Übersichtlichkeit halber nochmals programmiert.

### 3.2.3.4.7 Klassenmodul *cls_fehler*

Die Klasse *cls_fehler* ist für die Fehlerkontrollen zuständig, wobei jedoch auch zwei einfachere Methoden für Kontrollen, CheckZahl und CheckZahlInt im Formular programmiert wurden, da diese Methoden bei jedem Tastaturanschlag ausgeführt werden.

```
Public Function flex_reihenTest(flgNew As MSFlexGrid, Index As Integer, reiheNew As Integer) As Integer
```

Die Methode wird benötigt, um abzufragen, ob in einer Zeile in einer Tabelle bereits Werte stehen oder nicht. Dies ist nötig, um beispielsweise Fehler abzufangen, wenn in eine feststehende Zeile Werte geschrieben werden sollen. Übergeben wird das Objekt des Tabellensteuerelements und ein Index für die Tabelle, da diese in einem Steuerelementfeld vorliegt. Der letzte Übergabeparameter ist der Index der Reihe, in welche die Werte geschrieben werden sollen.

```
Public Function textboxtest(feldNew() As String, indexNew As Integer) As String()
```

Mit der Methode werden die Werte in den Textboxen überprüft und eventuelle Fehlermeldungen ausgegeben. Übergeben wird ein Array aus Stringwerten, mit dem Inhalt der Textboxen und ein Indexwert, der angibt, um welche Art von Werten es sich handelt. Innerhalb der Methode wird dann mit einer *Select Case*-Anweisung die entsprechende Überprüfung der Werte vorgenommen. Der Rückgabewert besteht aus einem Array mit den eingegebenen Werten, an welches in der letzten Zelle steht, ob eine Fehleingabe vorliegt oder nicht. Falls ja muss die Methode, in der die Funktion aufgerufen wurde beendet werden.

```
Public Function pruefeAbschnittslast(formNew As Object, indexNew As Integer, feldNew() As String) As
String()
```

Da die Eingaben von Abschnitts- und Punktlasten in verschiedenen Tabellen erfolgt, wird eine Methode benötigt, die überprüft, ob der Abschnitt auch tatsächlich existiert. Übergeben wird das Objekt der bestehenden Eingabeform, eine Indexnummer, die angibt, um welche Tabelle es sich handelt, und ein Array mit den eingegebenen Werten der Last.

### 3.2.3.5 Die Eingabemöglichkeit über Inputboxen des Betriebssystems

Wie in Kapitel 1.4.1.1 erwähnt, stellt das Betriebssystem Windows einige vordefinierte Funktionen oder auch Standarddialoge zur Verfügung, die immer wieder benötigt werden. Die beiden einfachsten Standarddialoge in VB oder VBA sind die Input-Box und die MsgBox. In einer Input-Box werden vom Benutzer Eingaben abgefragt, die über

Programmcode ausgewertet werden. Diese Möglichkeit der Eingabe und die Weiterverarbeitung wurde beispielsweise für die Funktionalität beim Auslesen von Profilwerten für den Stahlbau in MS Excel genutzt. Folgendes Programmlisting 19 aus der Datei *stahl6_9.xla* zeigt einen Codeabschnitt für die Nutzung dieser Möglichkeit. Somit wird die Programmierung einer eigenen Oberfläche vermieden und die Ausgabe der Werte kann in Strings, Integer und Double Werten erfolgen. Dis ist von Bedeutung, da in Excel die Einschränkung besteht, dass bei standardmäßig erzeugten Dialogfenstern nur ein Parameter ausgegeben oder zurückgegeben werden kann.

```
Zeile 1  Sub tafelWerte()            ' Beginn tafelWerte ()
Zeile 2  Dim dummy As String         ' Var. für Rückgabewert (Zeichenfolge) aus Inputbox
Zeile 3  Dim alphabet()              ' Datenfeld für Steuerung des Tabellenblatts
Zeile 4  Dim startzeile As Integer   ' Var. für den Rückgabewert (Integer) aus Inputbox
Zeile 5  Dim errNumber As String     ' Var für Fehlermeldung
Zeile 6  Dim returnValue As Integer  ' Var. für Rückgabewert aus Java Methode, keine weitere
Bedeutung
Zeile 7  Dim anzahlWerte As Integer  ' dient zum einlesen der Anzahl der ausgelesenen Werte
Zeile 8  Dim exitValue() As Double   ' Array 2 Felder für die Ergebnisse Biegeknicken u.
Biegedrillknicken
Zeile 9  Dim myRef As profTafel      ' Objektvar. für Java-Dll
Zeile 10 Dim ergebnis As String      ' Var. für Ergebnisausgabe in Excel
Zeile 11 Set myRef = New profTafel   ' Initialisierung der Java-Dll
```

Folgende Zeile lädt die Input Box von Bild 118 in den Speicher, nachdem die Funktion *tafelWerte()* über die Menüleiste für die *ED_TRA-Funktionen* aufgerufen wurde:

```
Zeile 12 dummy = InputBox("Bitte gebe Sie hier ihr Profil " + vbNewLine + "in Großbuchstaben und
ohne
                              Leerzeichen ein", "Profiltafel", "HEA300")
```

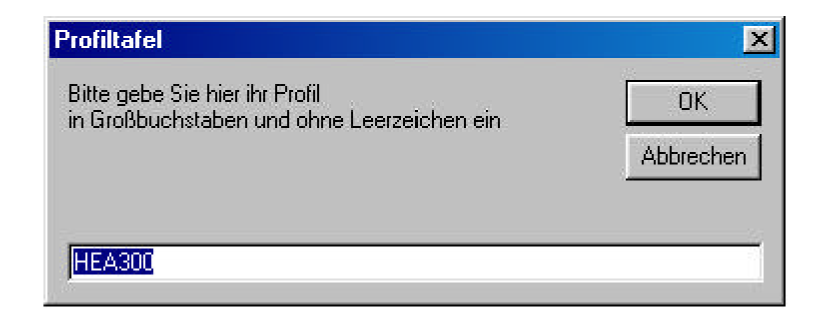



Bild 118. Input Box für die Eingabe eines Profils

```
‘ Auswertung des Rückgabewerts aus der Input Box. Wenn ““ eingegeben wurde -> Funktion abbrechen
Zeile 13 If dummy = "" Then
```

```
Zeile 14   GoTo Abbruch

Zeile 15 End If

' Einrichten des Datenfeld Alphabet für die Ergebnisausgabe beim Zugriff auf das Tabellenblatt
Zeile 16 alphabet() = Array("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", _
                    "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", _
                    "Z", "AA", "AB", "AC", "AD", "AE", "AF", "AG", "AH", "AI", "AJ", "AK")

' Abfrage, in welcher Zeile die Ausgabe der Ergebnisse beginnen soll
Zeile 17 startzeile = Val(InputBox("Bitte geben Sie die Zeile ein, " + vbNewLine + _
                    "in der die Ausgabe beginnen soll", "Profiltafel - Ausgabezeile", "1"))
Zeile 18 If startzeile = 0 Then

Zeile 19   GoTo Abbruch

Zeile 20 End If

Zeile 21 returnValue = myRef.profTafel(dummy)          ' Übergabe des Profils an Objekt myRef

Zeile 22 If returnValue < > 0 Then                     ' liest bei auftreten eines Fehlers die

Zeile 23   errNumber = myRef.getError(myRef.GetStatus())     ' zugehörige Meldung aus

Zeile 24   MsgBox errNumber                            ' Ausgabe der Fehlermeldung über die MsgBox

Zeile 25 Else

Zeile 26   anzahlWerte = myRef.getValueCount           ' Ermitteln der Anzahl der Rückgabewerte

Zeile 27   ReDim Preserve exitValue(anzahlWerte)             ' Einrichten einer Array für die
Rückgabewerte

Zeile 28   For i = 0 To anzahlWerte – 1                ' Einlesen aller Werte in exitValue in

Zeile 29        exitValue(i) = myRef.getWerte(i)        'einer Schleife

Zeile 30   Next i

Zeile 31 End If

' Prüfen, um welche Profilart es sich handelt und in entsprechende Ausgabe springen.
' Unterschieden werden 5 verschiedene Ausgabeformatierungen
Zeile 32 If Left(dummy, 3) = "IPE" Or Left(dummy, 2) = "HE" _        ' IPE o.HE-Profile
                    Or Left(dummy, 2) = "HL" _                         ' o. HL-Profile
                    Or Left(dummy, 2) = "HP" Or Left(dummy, 2) = "UB" _ ' HP oder UB-Profile
                    Or Left(dummy, 2) = "UC" Or Left(dummy, 1) = "W"        ' UC oder W-
Profile
                    Then

Zeile 33        ausgabe_1 myRef, dummy, exitValue(), startzeile, alphabet()

Zeile 34 ElseIf Left(dummy, 3) = "IPN" Or Left(dummy, 1) = "I" Then   ' IPN oder I-Profile

Zeile 35        ausgabe_2 myRef, dummy, exitValue(), startzeile, alphabet()

Zeile 36 ElseIf Left(dummy, 1) = "L" Then                                ' L-Profile
```

```
Zeile 37        ausgabe_3 myRef, dummy, exitValue(), startzeile, alphabet()
Zeile 38 ElseIf Left(dummy, 3) = "UPN" Then                        ‘ UPN-Profile
Zeile 39        ausgabe_4 myRef, dummy, exitValue(), startzeile, alphabet()
Zeile 40 End If
Zeile 41 Abbruch:                                   ’ Sprungmarke für
Abbruchbedingungen
Zeile 42 End Sub                                    ’ Ende tafelWerte
```

Programmlisting 19.

## 3.3 Benutzeroberflächen in Mathcad

Benutzeroberflächen in Mathcad existieren nur in Form von Mathcad Blättern, die als Eingabeoberflächen dienen. Diese sind in den unterschiedlichsten Formen denkbar und auch bereits vorhanden. Eine Liste von Herstellern und Hinweisen ist in Anhang D zu finden. Für Kapitel II-5 dieses Berichts wurden ebenfalls verschiedene Beispiele erstellt.

Die genaue Vorgehensweise bei der Einbindung der Funktionen, welche die Grundlage für die erstellten Mathcad-Arbeitsblätter bilden, werden in Kapitel 1.6.1.5 und Kapitel 1.6.2 erläutert.

# II-4 Handbücher für die Softwarebenutzung

## 4.1 Allgemeines

Dieser Teil des Berichts kann als Benutzerhandbuch für die erstellten Softwaremodule verstanden werden. Grundsätzliche Dinge, wie die Installation des Softwarebaukastens von der CD werden behandelt. Zudem werden die unterschiedlichen Benutzeroberflächen, die Eingabeparameter sowie die Ergebnisausgaben beschrieben.

## 4.2 Installation der Software

Für die Installation des Softwarebaukastens benötigen wird die im Rahen des Projekts erstellte CD benötigt. Da die CD außer dem Programm auch den gesamten Quellcode, sämtliche erstellten Textdateien, Hilfedateien usw. enthält, wurde der Strukturierung wegen das Setupprogramm in einem eigenen Ordner mit dem Namen ‚Setup' untergebracht.

Die Software installiert sich nicht selbst und muss daher vom Anwender entweder durch Doppelklicken auf die Setup-Anwendung in besagtem Ordner oder durch das Betätigen der Option *Öffnen* im Menü *Datei* der Menüleiste im Windows Explorer installiert werden. Der Anwender wird dann über unterschiedlich gestaltete Fenster durch die Installation begleitet, was im Folgenden erläutert wird.

Nach dem Start des Setup Programms erscheint der Eröffnungsbildschirm der Installationsroutine (Bild 119). Bevor die Installation gestartet wird, empfiehlt es sich alle anderen Anwendungen zu schließen.




Bild 119. Eröffnungsfenster der Installationsroutine des Softwarebaukastens

Nachdem der Button *Installieren* gedrückt wurde, erscheint das Dialogfenster für den zweiten Schritt der Installation, Bild 120. Hier wird der Pfad gewählt, in dem die Softwaremodule, Hilfedateien, Protokolldateien und die Uninstall Routine gespeichert wird. Dieser wird per Tastatur oder mit einem Doppelklick auf einen Ordner in der Auswahlbox in die Textbox auf der rechten Seite des Formulars eingetragen. Aufgrund der Übersichtlichkeit empfiehlt es sich einen eigenen Ordner für den Softwarebaukasten zu wählen.

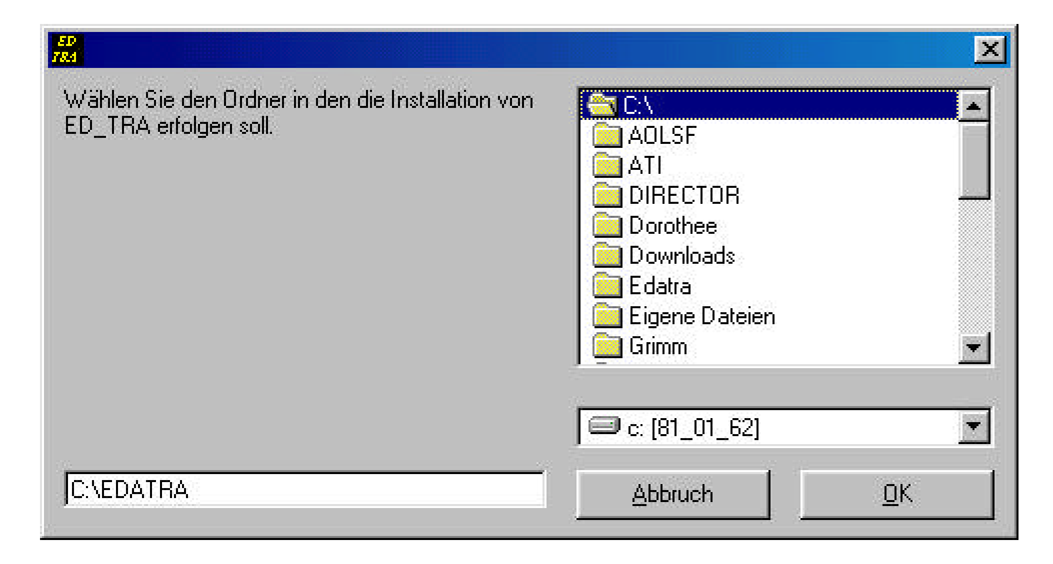



Bild 120. Schritt 2 der Installationsroutine für den Softwarebaukasten

Die COM-Komponenten oder Softwaremodule mit den Berechnungsfunktionen müssen in diesem Ordner registriert werden. Dies wird von der Installationsroutine übernommen. Nach dem Kopieren und Registrieren der Dateien erscheint der nächste Schritt der Installation, wie

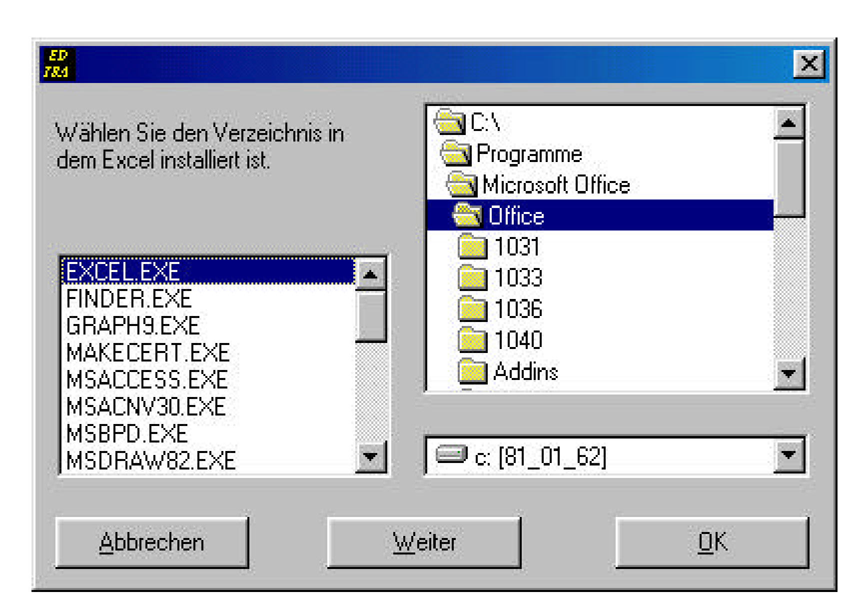



Bild 121. Schritt 3 der Installation des Softwarebaukastens

in Bild 121 zu sehen ist. In diesem Dialogfenster wird der Pfad ausgewählt, in dem die Excel Anwendung gespeichert ist. *Excel.exe* muss hierbei in der Listbox auf der linken Fensterseite angezeigt werden. Nachdem dann auf *OK* geklickt wurde werden verschiedene *AddIns*, bzw. Dateien mit der Endung *.xla*, in das Startverzeichnis von Excel kopiert. Ab nun ist die Funktionalität des Softwarebaukastens in Excel verfügbar. Falls die Installation der *AddIns* nicht gewünscht wird, kann dieser Schritt übergangen werden, indem auf Weiter *geklickt* wird.

Anschließend wird das Dialogfenster für die Installation der Dateien für die Mathcad-Anwendung geöffnet (Bild 122). Die Vorgehensweise ist gleich wie in Schritt 3. Auf der

linken Seite muss die Datei *Mathcad.exe* in der Listbox erscheinen, dann kann die Installation mit *OK* bestätigt werden




Bild 122. Schritt 4 der Installation des Softwarebaukastens

## 4.3 Deinstallieren der Software

Während der Installation wurde im Installationspfad des Softwarebaukastens, siehe Bild 120, die Anwendung *uninst.exe* kopiert. Nach dem Starten der Anwendung wird der Assistent für die Deinstallation gestartet, wie Bild 123 zeigt. Dieser besteht lediglich aus einem Dialogfenster, in dem über den Button *ED_TRA entfernen* die Deinstallation ausgeführt wird.

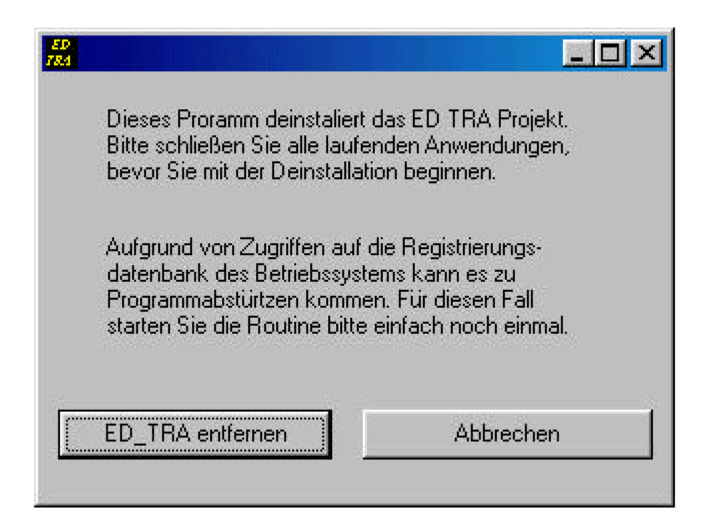



Bild 123. Dialogfenster der Deinstallationsroutine des Softwarebaukastens

Nach der erfolgreichen Deinstallation erscheint eine Meldung, dass die Deinstallation erfolgreich abgeschlossen wurde.

Bei verschiedenen Versuchen während der Erstellung der Anwendung kam es auf manchen Rechnern zu Laufzeitfehlern bei der Deinstallation des Softwarebaukastens, was mit dem

Zugriff auf die Registrierungsdatenbank des Windows Betriebssystems zusammenhängt. Falls dies auch auf Ihrem Rechner der Fall sein sollte, kann das Entfernen mit einem erneuten Start der Anwendung *uninst.exe* problemlos wiederholt werden.

Aufgrund der Datensicherung werden nicht alle Dateien, die in den Installationsordner aus Schritt 2 auf die Festplatte kopiert wurden, wieder entfernt. Dies gilt beispielsweise für die Hilfedateien für den Softwarebaukasten oder für Exceldateien, in denen Änderungen vom Benutzer gespeichert sein könnten. Diese müssen, falls gewünscht, von Hand entfernt werden.

## 4.4 Durchlaufträger

### 4.4.1 Allgemeines

Dieses Kapitel stellt die Hilfe zur Anwendung des Durchlaufträger–Moduls dar. Der Anwender erhält hierbei eine schrittweise Anleitung zur Ein- und Ausgabe für einen Durchlaufträger.

### 4.4.2 Verwendung der Softwarekomponenten in MS-Excel

#### 4.4.2.1 Aufruf der Eingabe

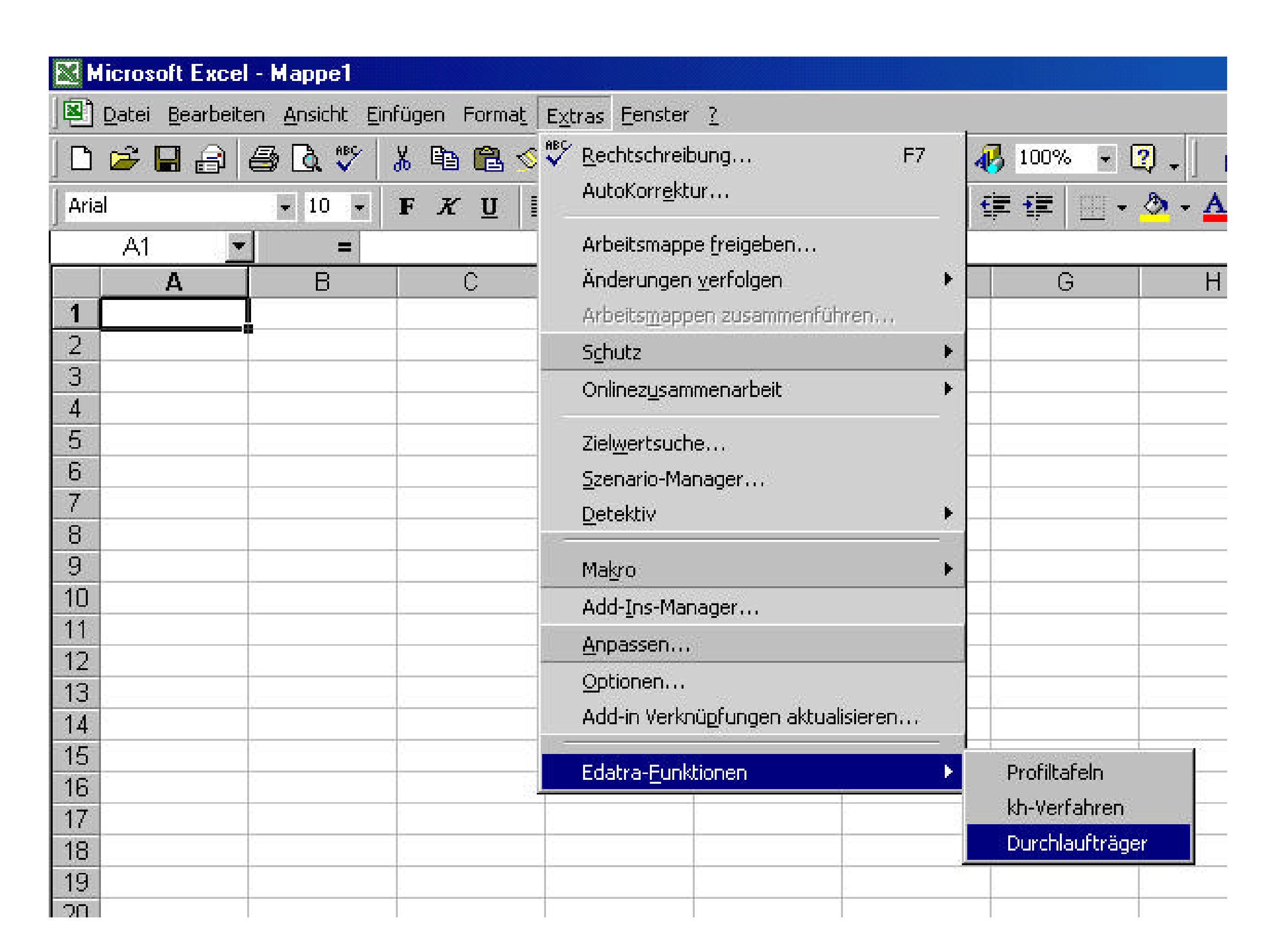



Bild 124. Benutzerdefinierter Menüpunkt „Edatra“ in der Excel Anwendung

Nach der Installation des Softwarebaukastens für Excel, siehe Kapitel 4.2, und dem Öffnen einer neuen Arbeitsmappe in Microsoft Excel befindet sich ein benutzerdefinierter Menüpunkt „Edatra“ in der Menüleiste „Extras“, siehe Bild 124.

Über diesen Menüpunkt stehen mehrere Statikfunktionen zur Verfügung. Durch Auswahl von *Durchlaufträger* wird die Eingabemaske für die Berechnung von Durchlaufträgern in den Speicher geladen. Die Eingabemaske ist im unteren Bild zu sehen.

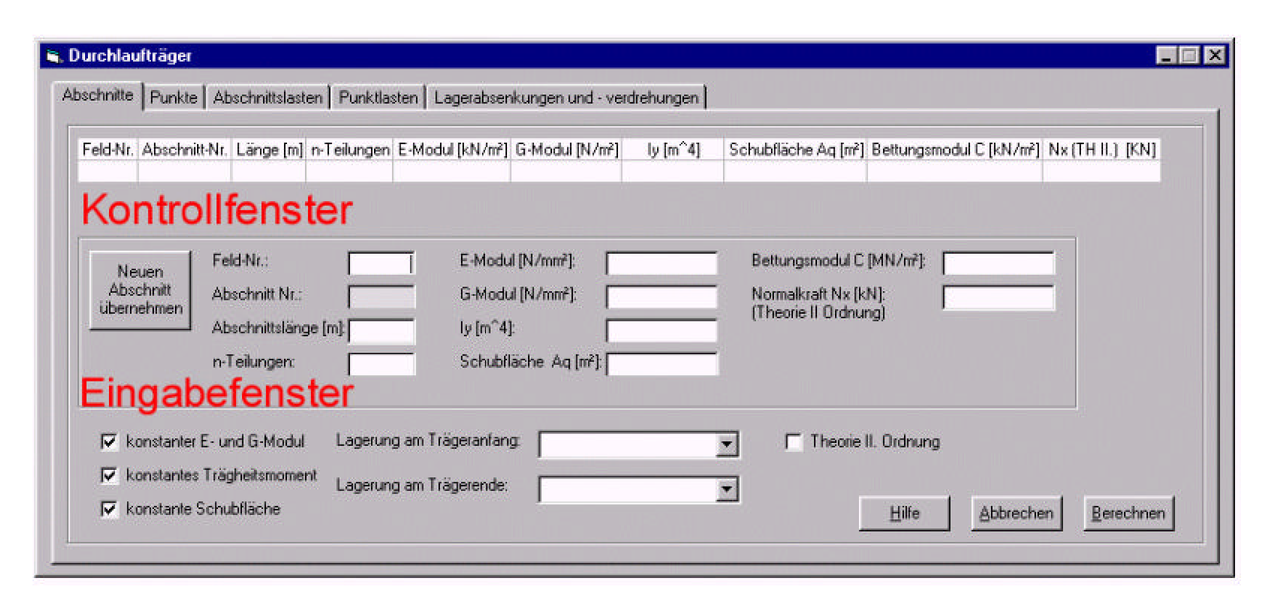



Bild 125. Eingabemaske für die Berechnung von Durchlaufträgern

### 4.4.2.2 Eingabe der Daten

#### Eingabe der Abschnitte

Die Eingabe des Systems erfolgt abschnittweise. Die Aufteilung des Systems in Abschnitte kann aus Bild 126 entnommen werden.

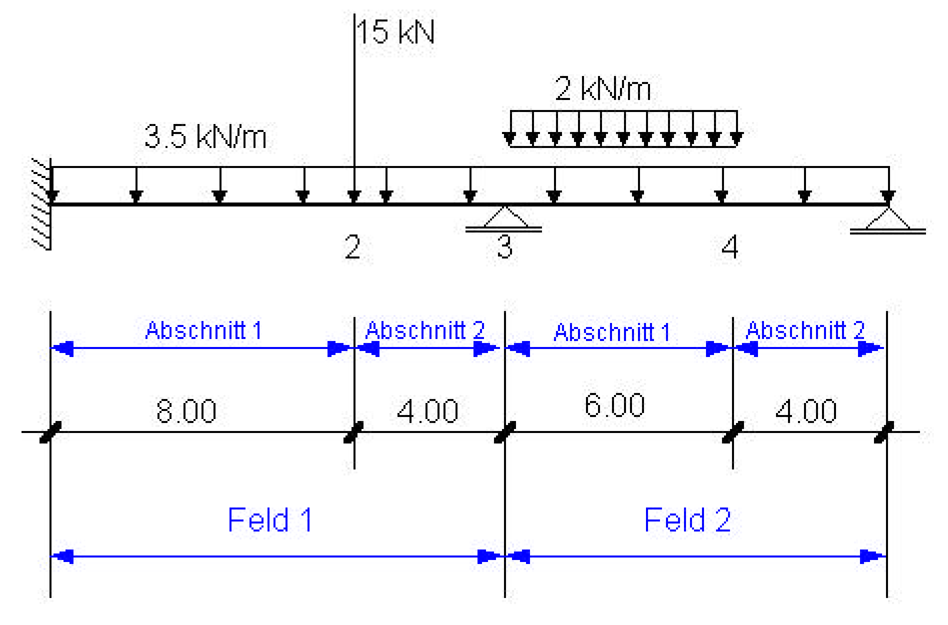



Bild 126. Feldeinteilung für einen Durchlaufträger

Die Eingabe erfolgt immer von links nach rechts. Nach der Eingabe eines Abschnittes muss der Button *„Neuen Abschnitt übernehmen“* betätigt werden. Die Werte erscheinen dann im Kontrollfenster und es kann ein neuer Abschnitt eingeben werden. Wird die Definition für einen Abschnitt nicht in die Tabelle, bzw. in das Kontrollfenster übernommen, bleibt diese bei der Berechnung unberücksichtigt. Dies gilt im übrigen auch für alle anderen Tabellenblätter.

**Eingabe der Systemwerte**

Im Folgenden werden die Eingabewerte für einen Abschnitt näher erläutert:

| | |
|---|---|
| Feld – Nummer: | Feldgrenzen werden definiert durch Auflager |
| Abschnittslänge [m]: | Abschnittsgrenzen werden definiert durch Federn, sprunghafte Belastungsänderung bei Streckeneinwirkungen, Punkteinwirkungen |
| n-Teilungen: | Über die Eingabe der n-Teilungen werden die Ergebnisausgabepunkte für die Momente, die Querkräfte sowie die Durchbiegung festgelegt. |
| E-Modul [N/mm$^2$]: | Elastizitätsmodul |
| Iy [cm$^4$]: | Trägheitsmoment. Die Eingabe der Biegesteifigkeit Elastizitätsmodul E-Modul * Trägheitsmoment Iy ist für die Berechnung immer erforderlich. |
| G-Modul [N/mm$^2$]: | Schubmodul |
| $A_Q$ [cm$^2$]: | Schubfläche. Bei dem Sonderfall, dass die Querkraftverformung vernachlässigt werden soll, kann die Schubsteifigkeit Schubmodul G-Modul * Schubfläche $A_Q$ zu Null gesetzt werden. |
| C [MN/m$^2$]: | Bettungsmodul für elastische Bettung |
| N [kN] Th. II Ordnung: | Normalkraft für die Berechnung nach Theorie II Ordnung |

Tabelle 12. Systemwerte für einen Durchlaufträger

Achtung!
Bei der Eingabe eines Bettungsmoduls sowie einer Normalkraft für die Berechnung nach Theorie II Ordnung, muss die Schubsteifigkeit Schubmodul *G-Modul* * Schubfläche $A_Q$ immer eingegeben werden.

**Nachträgliches Eingaben oder Löschen von Abschnitten**

Um einen Abschnitt nachträglich einzufügen, muss im Kontrollfenster die Zeile markiert werden, an dem der neue Abschnitt eingefügt werden soll. Über das Kontextmenü (rechte Maustaste) kann die Option Abschnitt einfügen gewählt werden und man gelangt wieder ins Eingabefenster. Das Löschen eines Abschnittes erfolgt analog.

## Eingabewerte für das Gesamtsystems

Wenn über den gesamten Trägerverlauf keine Material- und Querschnittsänderung erfolgt, kann dies im unteren Teil der Eingabemaske gewählt werden. Es ist somit nur eine Eingabe der Biegesteifigkeit (Elastizitätsmodul E-Modul * Trägheitsmoment Iy) und der Schubsteifigkeit (Schubmodul G-Modul * Schubfläche $A_{Q)}$ beim ersten Abschnitt erforderlich.

Für die Lagerung des gesamten Trägers muss die entsprechende Lagerungsart am Trägeranfang und am Trägerende, wie in Bild 127 dargestellt, gewählt werden.

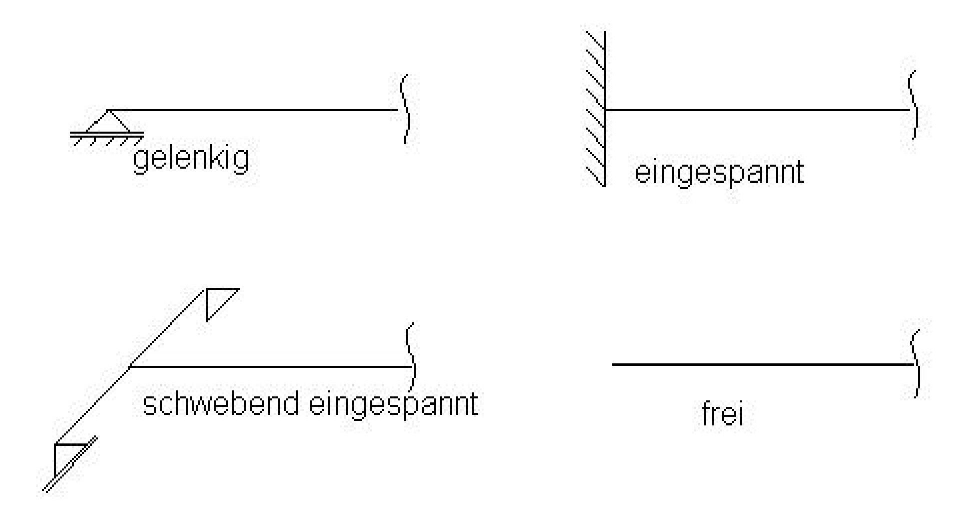



Bild 127. Lagerungsarten am Trägeranfang und -ende

## Eingabe der Punkte

Um die Definition für Punkte einzugeben muss zum Tabellenblatt[1] *Punkte* gewechselt werden. Dies geschieht durch einfaches anklicken des TabnStrips auf der Benutzeroberfläche, siehe Bild 128.

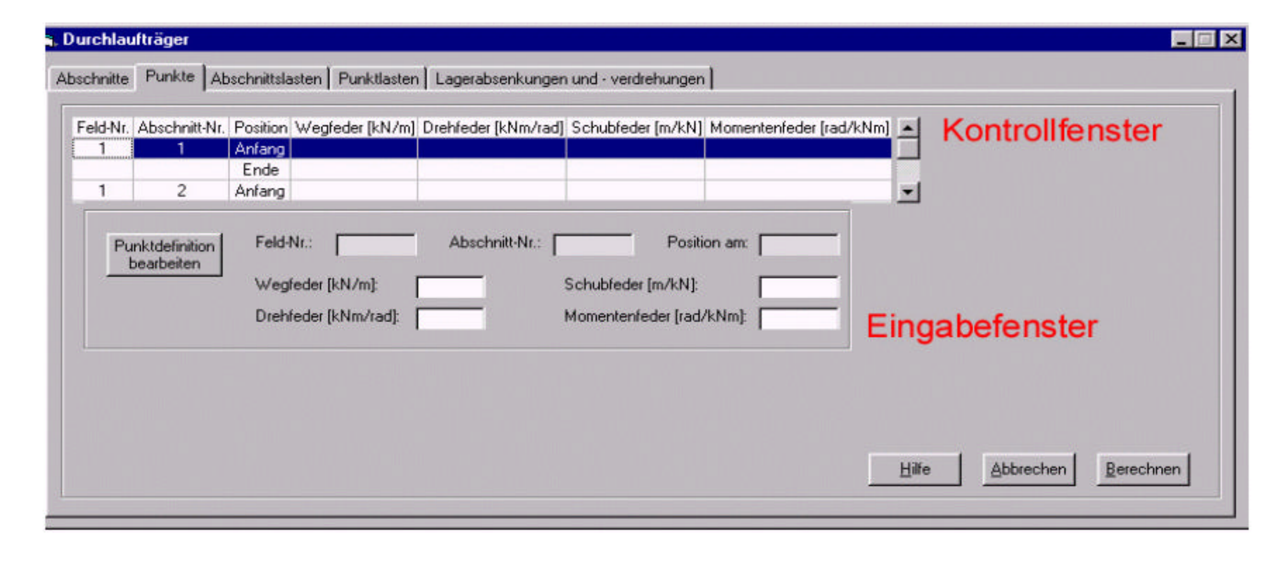



Bild 128. Tabellenblatt für die Eingabe der Punkte

[1] Tabellenblätter - TabStrip in VB. Ein Tabellenblatt Steuerelement. Wird im Folgenden auch TabStrip genannt.

Die Definition eines Punktes ergibt sich hauptsächlich aus den verschiedenen Möglichkeiten für Federn, die hier definiert werden können (Bild 129).

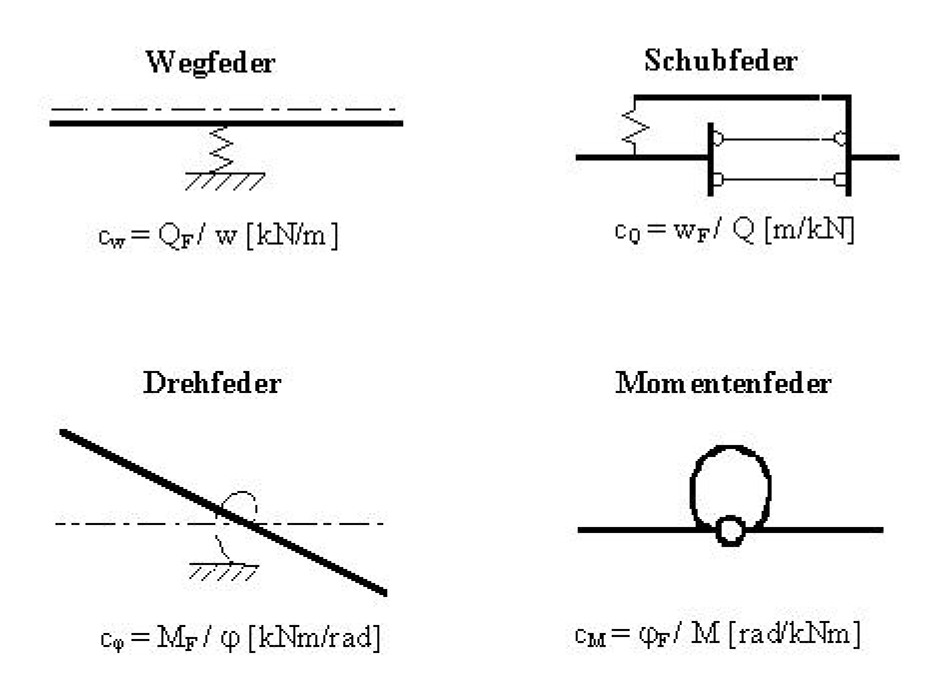



Bild 129. Definitionarten der Federn für Punkte

Die Eingabesyntax für die Eingabe der Feld-Nr., der Abschnitts-Nr., mit der jeweiligen Position (siehe Bild 130) erfolgt durch das Markieren der Zeile, an der die Feder eingegeben werden soll. Mit dem Button „*Punktdefinition bearbeiten*“ gelangt man dann ins Eingabefenster. Nach der Eingabe der Feder muss der Button „*Punktdefinition übernehmen*“ betätigt werden. Die Werte erscheinen im Kontrollfenster und es kann eine neue Feder eingeben werden.

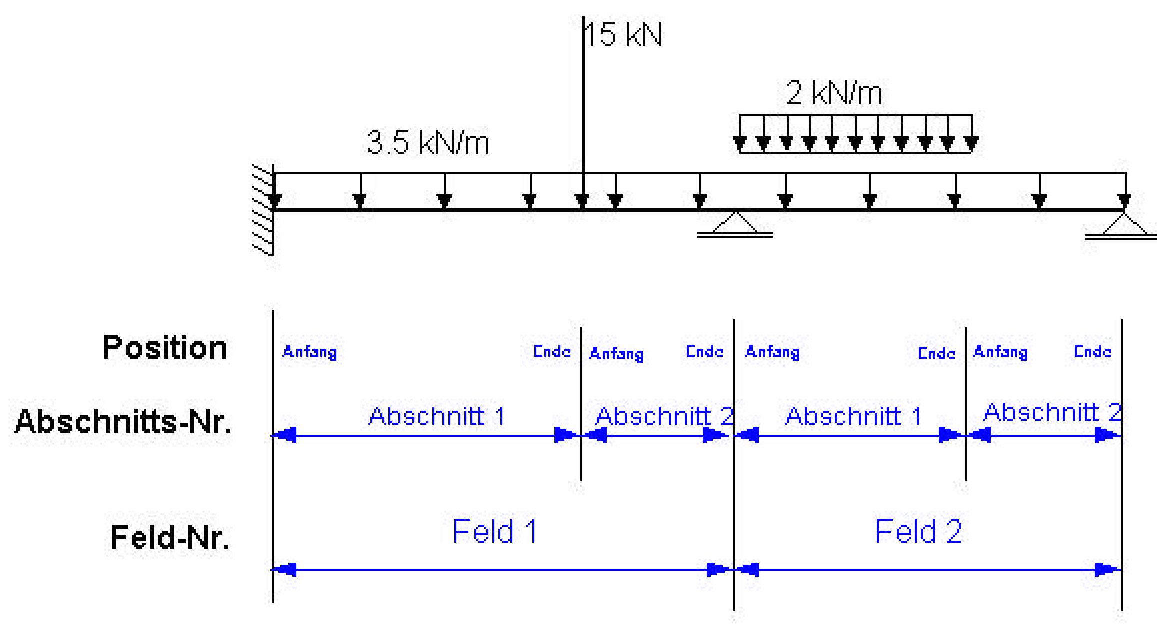



Bild 130. Eingabesyntax für die Punkte

Aufgrund der Eingabesyntax, die Feldweise von links nach rechts erfolgt, ergibt sich die Notwendigkeit der Eingabe einer Position, an der Federdefinitionen eingegeben werden. Hieraus ergeben sich zwar Redundanzen in der Tabelle, jedoch werden diese programmintern verwaltet. Vom Benutzer wird lediglich an einer Stelle, also beispielsweise bei Feld 1 am Ende oder bei Feld 2 am Anfang die Definition eingegeben. Die Eingabe erfolgt also nicht zweifach.

### Eingabe der Abschnittslasten

Die Eingabesyntax für die Abschnittslasten ist analog der Eingabe für die Definition der Abschnitte.

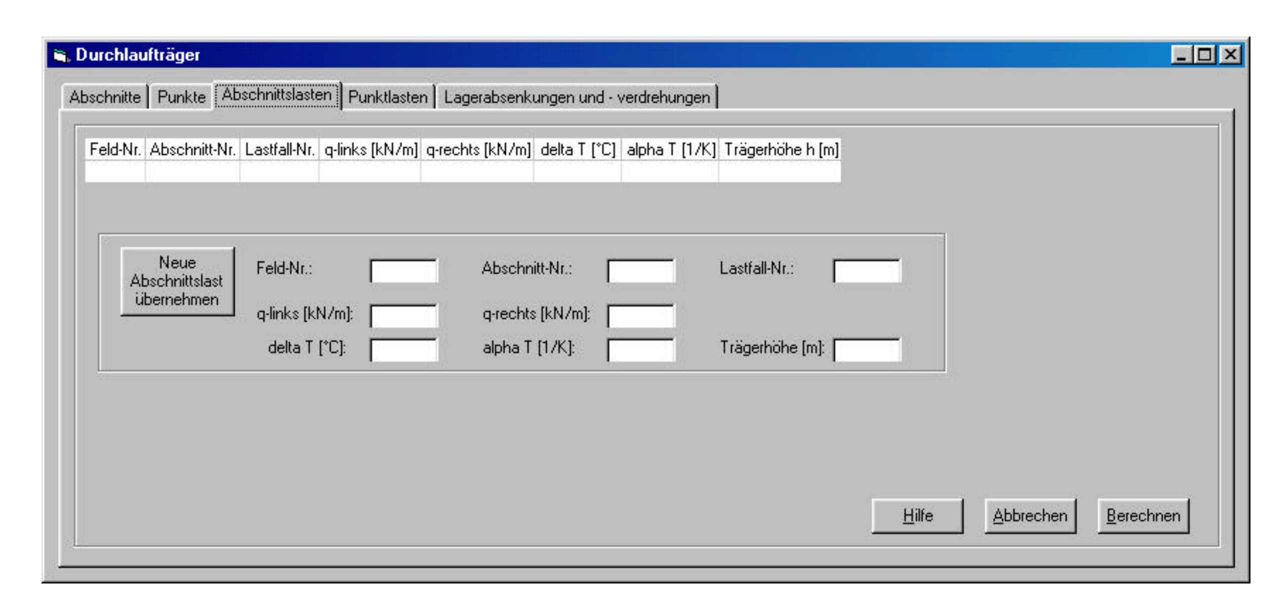



Bild 131. Tabellenblatt für die Eingabe der Abschnittslasten

Tabelle 13 zeigt die Eingabewerte für die Definition einer Abschnittslast. Bild 132 zeigt eine trapezförmige Abschnittslast. Gleichförmige Abschnittslasten werden durch Eingabe gleicher Lastwerte am Anfang und Ende definiert. Nach der Eingabe der Last muss der Button *„Neue Abschnittslast übernehmen“* betätigt werden, um die Definition der Last für die Berechnung zu übernehmen.

| **Eingabe der Abschnittslasten** | |
|---|---|
| Feld – Nr. | Nummer des Feldes in der die Last angreift |
| Abschnitts – Nr. | Nummer des Abschnittes in der die Last angreift |
| Lastfall – Nr. | Nummer des Lastfalls<br>Lastfall Nr.1 beinhaltet die ständigen Lasten<br>Lastfall Nr.2 und folgende Lastfälle beinhalten Verkehrslasten |
| **Gleichlast / Trapezlast** | |
| q - links [kN/m] | Belastungsgröße am Abschnittsanfang, siehe Bild 132 |
| q - rechts [kN/m] | Belastungsgröße am Abschnittsende, siehe Bild 132 |

| **Temperaturlasten** | |
|---|---|
| Delta T [° C] | Temperaturänderung an der unteren bzw. oberen Stabseite (untere Stabseite = Seite, an der sich die Zugfaser befindet) |
| Alpha T [1/K] | Wärmedehnzahl der verschiedenen Materialien |
| Trägerhöhe h [m] | Querschnittshöhe |

Tabelle 13. Eingabeparameter der Abschnittslasten für einen Durchlaufträger

Um eine bereits eingegebene Abschnittslast zu ändern, muss die Zeile, in der die Abschnittslast definiert ist, in der Tabelle markiert werden und über den Button *„Abschnittslast bearbeiten“* für die Bearbeitung in die Textfelder übernommen werden.

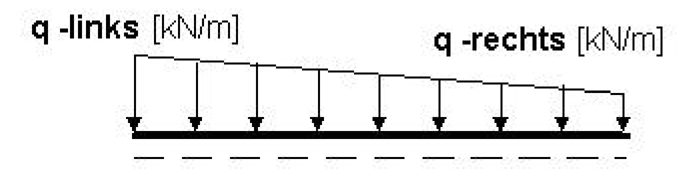



Bild 132. Abschnittslasten für einen Durchlaufträger

In der Tabelle erscheinen die Abschnittslasten primär sortiert nach Lastfall und sekundär nach der Feldnummer. Dennoch wurde die Reihenfolge der Werte aufgrund des einheitlichen Layouts in allen TabStrips beibehalten. So können die einzelnen Abschnittslasten nach der primären und sekundären Sortierung immer noch durcheinander in der Tabelle erscheinen, was die Abschnittsnummer betrifft.

## Eingabe der Punktlasten

Die Vorgehensweise bei der Eingabe der Punktlasten ist analog der Eingabe für die Abschnittslasten.

Durchlaufträger

Abschnitte | Punkte | Abschnittslasten | Punktlasten | Lagerabsenkungen und - verdrehungen

| Feld -Nr. | Abschnitt-Nr. | Position | Lastfall-Nr. | Fz [kN] | My [kNm] | delta w [m] | delta-Phi [rad] |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Anfang | 1 | 1.00 | - | - | - |

Neue punktuelle Last übernehmen

Feld-Nr.: Abschnitt-Nr.: Position:

Lastfall-Nr.: F [kN]: delta W [m]:

M [kNm]: delta Phi [rad]:

Hilfe Abbrechen Berechnen

Bild 133. Tabellenblatt für die Eingabe der Abschnittslasten

In folgender Tabelle sind wieder die Eingabeparameter mit ihren Dimensionen aufgeführt.

| | |
|---|---|
| Feld – Nr. | Nummer des Feldes in der die Last angreift |
| Abschnitts – Nr. | Nummer des Abschnittes in der die Last angreift |
| Position | Position des Lastangriffs (Bild 130). Die Eingabe der Last erfolgt entweder am Anfang oder am Ende eines Abschnittes |
| Fz [kN]: | Einzellast, positiv gemäss Bild 134 |
| My [kNm]: | Einzelmoment, positiv gemäss Bild 134 |
| Delta w [m]: | Eingeprägte Einzeleinwirkungen Relativverschiebungen, positiv gemäss Bild 134 |
| Delta φ [rad]: | Eingeprägte Einzeleinwirkungen Knickwinkel, positiv gemäss Bild 134 |

Tabelle 14. Eingabeparameter der Punktlasten für einen Durchlaufträger

Nachfolgendes Bild 134 zeigt die Definition der positiven Richtung der Lasten. Auch bei den Punktlasten verhält es sich so, dass sie entweder am Ende eines Abschnitts eingegeben wird oder am Anfang. Eine Auswertung der Eingabe erfolgt dann wieder programmintern.

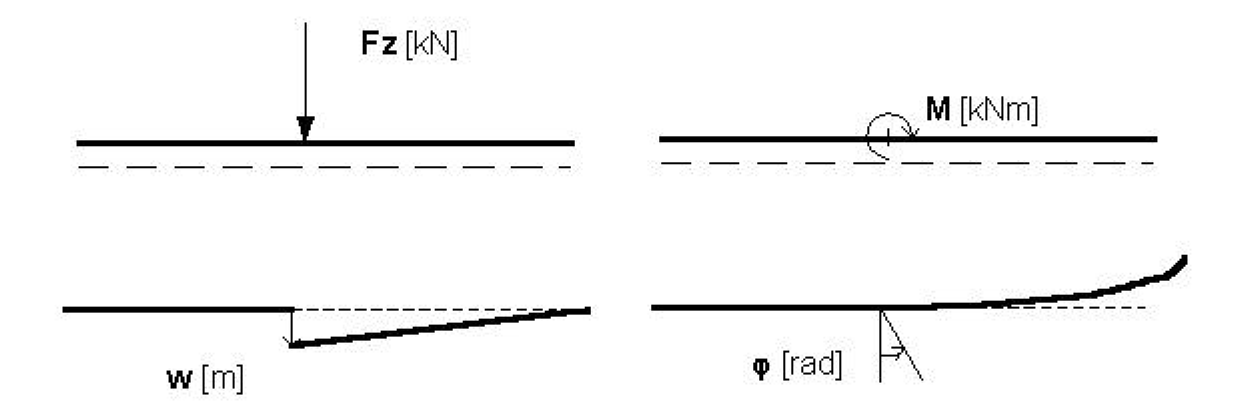



Bild 134. Definition der positiven Richtung der Lasten für einen Durchlaufträger

## Eingabe der Lagerabsenkungen und –verdrehungen

Die Eingabesyntax ist wieder Analog der Eingabe für die Punktlasten

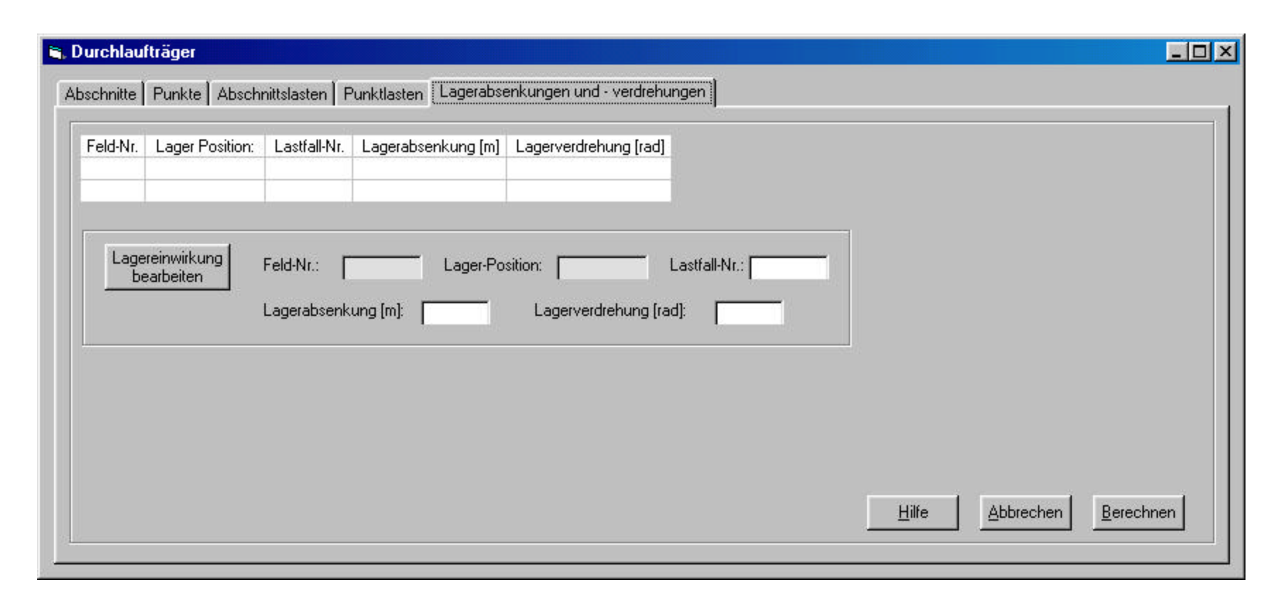



Bild 135. Tabellenblatt für die Eingabe der Lagerverdrehung und Lagerverschiebung

| Eingabe der Werte | |
|---|---|
| Lastfall - Nr. | Nummer des Lastfalls. Es muss immer ein eigenständiger Lastfall für eine Lagerabsenkung bzw. Lagerverdrehung definiert werden. |
| Lagerabsenkungen [m] | Verschiebung am Lager, positiv gemäß Bild 136 |
| Lagerverdrehung [rad ] | Verdrehung der Einspannung am Lager, positiv im Uhrzeigersinn siehe Bild 136 |

Tabelle 15. Eingabeparameter für die Lagerverdrehung und Lagerverschiebung

Lagerverschiebung

Lagerverdrehung

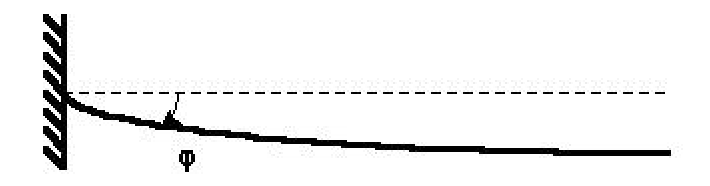



Bild 136. Definition der positiven Lagerverschiebung und -verdrehung

#### 4.4.2.3 Ausgabe der Eingabe- und Ergebnisdaten

Nach Beendigung der Eingabe werden über den Button „Berechnen“ die Ergebnisse berechnet. Für die Ausgabe der Daten erscheint die Abfrage in welcher Zeile einer Excel-Arbeitsmappe die Ausgabe beginnen soll. Diese Abfrage bietet die Möglichkeit in eine bereits mit Daten beschriebene Arbeitsmappe die Ausgabe hinein zu schreiben. Hierbei werden im Excelblatt, das bereit Werte enthält, diese nicht überschrieben. Für jede Ergebniszeile wird eine Zeile zum Blatt hinzugefügt, somit werden Zellen, welche vor der Berechnung bereits Werte enthielten, nach unten verschoben.

### 4.4.3 Verwendung der Funktionen in Mathcad

Für die Verwendung der Softwarekomponente in Mathcad wurden aufgrund der Schnittstelle, die hier lediglich für C, bzw. C++ vorhanden ist, was auch an anderer Stelle bereits erwähnt wurde, mehrere Funktionen programmiert.

Nachstehende Liste zeigt die Funktionen, die im Programm Mathcad zur Verfügung stehen.

DLT_A_MinMax

DLT_A_proLF

DLT_M_MinMax

DLT_M_proLF

DLT_Ma_MinMax

DLT_Ma_proLF

DLT_MaxM

DLT_Q_MinMax

DLT_Q_proLF

DLT_W_MinMax

DLT_W_proLF

Die Eingabe der Berechnungsparameter für die Funktionen werden anhand eines konkreten Beispiels nachfolgend näher erläutert.

## Beispiel für die Eingabe eines Durchlaufträgers

### Systemskizze:

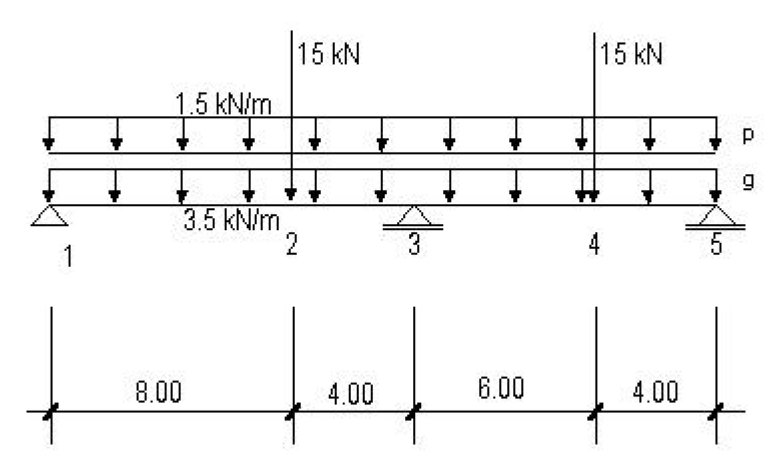



Bild 137. Systemskizze des Beispiels

### Systemwerte:

**Feldlängen [m]**

$L_1 := 12.00$

$L_2 := 10.00$

**Abschnittslängen [m]**

$l_{a1} := 8.00$

$l_{a2} := 4.00$

$l_{a3} := 6.00$

$l_{a4} := 4.00$

**Querschnittswerte**
**E-Modul [kN/m^2]**
**Trägheitsmodul [m^4]**

Feld1

$E_1 := 30500000$

$I_1 := 0.001898437$

$EI_1 := E_1 \cdot I_1$ $\qquad EI_1 = 57902.33$

Feld2

$E_2 := 3050000$

$I_2 := 0.001898437$

$EI_2 := E_2 \cdot I_2$ $\qquad EI_2 = 5790.23$

### Zusammenstellung der Belastung:

**Streckeneinwirkungen [kN/m]**
**Punkteinwirkungen [kN]**

| | | |
|---|---|---|
| ständige Einwirkungen | aus Position Platte | $g := 3.50$ |
| veränderliche Einwirkungen | aus Position Platte | $p := 1.50$ |
| | aus Position Stütze | $P := 15.00$ |

Die Eingabewerte für die Funktionen:

Zeilenweise Eingabe der Abschnitte
(Feld_Nummer, Abschnitts_Nummer, Abschnitts_Länge[m]

$$Ab := \begin{pmatrix} 1 & 1 & l_{a1} \\ 1 & 2 & l_{a2} \\ 2 & 1 & l_{a3} \\ 2 & 2 & l_{a4} \end{pmatrix}$$

Abschnitts-Länge ( Ab_Länge[m] )
Abschnitts-Nummer ( Ab_Nr )
Feld-Nummer ( Feld_Nr )

Feldweise Eingabe der Querschnittsdefinitionen [kN*m^2]

$$EI := \begin{pmatrix} EI_1 & EI_2 \end{pmatrix}$$

Feld 1, Feld 2

Feldweise Eingabe des Bettungsmodul [kN/m^3]

$$Cs := \begin{pmatrix} 0 & 0 \end{pmatrix}$$

Feld 1, Feld 2

Feldweise Eingabe der Normalkraft Th.II Ordnung [kN]. Wird N = 0 erfolgt die Berechnung nach Theorie I. Ordnung

$$N := \begin{pmatrix} 0 & 0 \end{pmatrix}$$

Feld 1, Feld 2

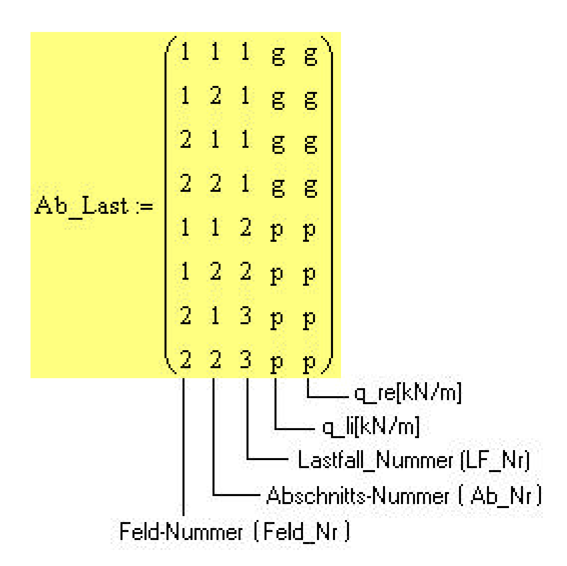
Lagerdefinition am linken Trägerrand

$$Lli := 1$$

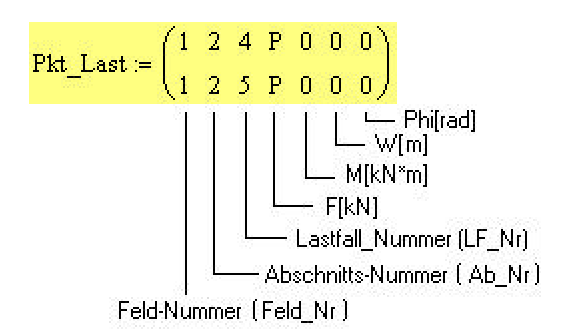
Lagerdefinition am rechten Trägerrand

$$Lre := 1$$

Zeilenweise Eingabe der Abschnittslasten
(Feld_Nummer, Abschnitts_Nummer, Lastfall_Nummer, q_li [kN/m], q_re [kN/m])




Zeilenweise Eingabe der Abschnittslasten
(Feld_Nummer, Abschnitts_Nummer, Lastfall_Nummer, F[kN], M[kN*m], W[m], Phi[rad])




**DLT_A_MinMax**

Die Funktion liefert die minimalen und maximalen Auflagerkräfte lastfallweise über den Trägerverlauf in einer n*m Matrix.

Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Eingabe:

Ab : Eine n*m Matrix. Zeilenweise Eingabe der Abschnitte (Feld_Nr,Ab_Nr,Ab_Länge[m] )

EI : Feldweise Eingabe von E[kN/m^2] * I[m^4] in einem Vektor

Cs : Feldweise Eingabe des Bettungsmoduls[kN/m^3] in einem Vektor

N : Feldweise Eingabe der N[kN] nach Th II Ordnung in einem Vektor

Lli: Lagerdefinition am linken Trägerrand

1 - gelenkig 2 - eingespannt
3 – frei 4 - schwebend eingespannt

Lre: Lagerdefinition am rechten Trägerrand

1 - gelenkig 2 - eingespannt
3 – frei 4 - schwebend eingespannt

Ab_Last: Eine n*m Matrix. Zeilenweise Eingabe der Abschnittslasten (Feld_Nr, Ab_Nr, LF_Nr,q_li[kN/m],q_re[kN/m])

Pkt_Last: Eine n*m Matrix. Zeilenweise Eingabe der Punktlasten (Feld_Nr, Ab_Nr,LF_Nr,F[kN],M[kN*m],W[m],Phi[rad])

x: Länge der Ausgabeabschnitte. Hier ist die Unterteilung gemeint, an deren Stellen die Ergebniswerte berechnet werden

**DLT_A_proLF**

Die Funktion liefert die Auflagerkräfte Lastfallweise über den Trägerverlauf in einer n*m Matrix.
Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_M_MinMax**

Die Funktion liefert die minimalen und maximalen Momente lastfallweise über den Trägerverlauf in einer n*m Matrix.
Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_M_proLF**

Die Funktion liefert die Momente lastfallweise über den Trägerverlauf in einer n*m Matrix.
Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_Ma_MinMax**

Die Funktion liefert die minimalen und maximalen Auflagermomente Lastfallweise über den Trägerverlauf in einer n*m Matrix
Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)
Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_Ma_proLF**

Die Funktion liefert die Auflagermomente lastfallweise über den Trägerverlauf in einer n*m Matrix.

Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_MaxM(Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)**

Die Funktion liefert maximale Feldmomente über den Trägerverlauf in einer n*m Matrix.

Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_Q_MinMax**

Die Funktion liefert die minimale und maximale Quekräfte lastfallweise über den Trägerverlauf in einer n*m Matrix.

Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_Q_proLF(Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)**

Die Funktion liefert die Momente Auflagerkräfte über den Trägerverlauf in einer n*m Matrix.

Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_W_MinMax(Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)**

Die Funktion liefert die minimale und maximale Durchbiegung Lastfallweise über den Trägerverlauf in einer n*m Matrix.

Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax.*

**DLT_W_proLF(Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)**

Die Funktion liefert die Durchbiegung lastfallweise über den Trägerverlauf in einer n*m Matrix.

Eingabeparameter: (Ab,EI,Cs,N,LLi,LRe,AbLast,PktLast,X)

Die Syntax der Parameterübergabe ist in Anzahl und Reihenfolge analog der Funktion *DLT_A_MinMax*.

## 4.5 Fachwerk

### 4.5.1 Allgemeines

Die Komponenten und Dateien zur Berechnung von Fachwerken wurden innerhalb eines studentischen Projekts[2], welches an dieses Forschungsprojekt angeschlossen war, umgesetzt.

### 4.5.2 Verwendung des Softwaremoduls in Excel[3]

Dieses Kapitel ist die Anwenderdokumentation für die Berechnung von Fachwerken in Excel. Der Prozess der Berechnung eines Fachwerks in Excel unter Verwendung der bereits beschriebenen Java–COM-Dll ist in 3 Schritte aufgeteilt. Jeder der nachfolgend beschriebenen Schritte wird auf der Oberfläche des Arbeitsblattes „Tragwerk" in der Excel–Datei „Fachwerk.xls" durch einen Button mit gleichnamiger Aufschrift repräsentiert. Die Datei „Fachwerk.xls" befindet sich im Installationsordner von Edatra, siehe Kapitel 4.2.

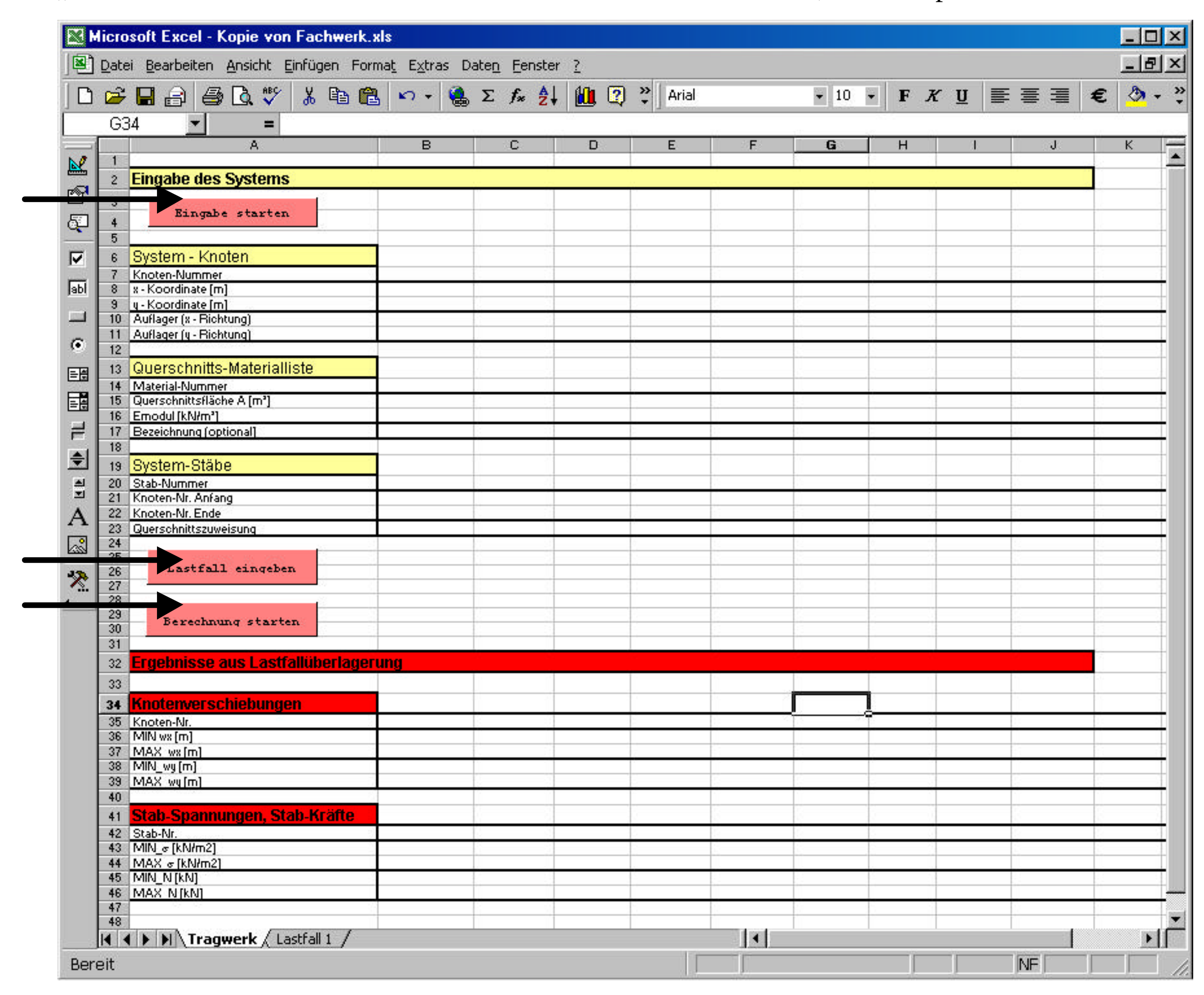



Bild 138. Das Arbeitsblatt „Tragwerk" (Ausgangszustand) der Excel-Datei Fachwerk.xls

[2] Projekt EDATRA im Fachbereich Wirtschaftsinformatik an der Fachhochschule Konstanz Wintersemester 2000/2001

[3] Verwendet wurde hier das Packet Office 2000 der Firma Microsoft

### 4.5.2.2 Der Button Eingabe starten

Beim Klicken auf den Button „Eingabe starten“ erscheint als erstes das in Bild 139 dargestellte Informationsbild mit dem Hinweis auf die positive x bzw. y Kräfterichtung, die der Berechnungen zugrunde liegt und der die Eingaben entsprechen müssen, um korrekte Ergebnisse zu erhalten.

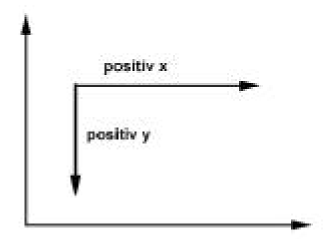



Bild 139. Positiver x bzw. y Kräfteverlauf

Danach werden alle alten Eingaben gelöscht (System – Knoten, Querschnitts – Materialliste, System – Stäbe und die Arbeitsblätter alter Lastfälle). Arbeitsblatt „Lastfall 1“ wird nicht gelöscht, da dieser Lastfall in jedem Fall eingegeben werden muss und somit immer vorhanden ist.

Nun können die System–Knoten eingegeben werden. Die erscheinende Dialogreihe soll die Eingabe vereinfachen. Über die Dialoge kann die x–Koordinate, die y–Koordinate und die Festhaltung in x-, bzw. y-Richtung jedes Knotens eingegeben werden. Sobald alle Knoten eingegeben wurden wird durch einen Klick auf „Abbrechen“ die Knoteneingabe beendet. Es erscheint die nächste Dialogreihe, die zur Eingabe der Materialien auffordert, die wiederum durch klicken auf „Abbrechen“ beendet werden kann. Die Eingabe der Systemstäbe funktioniert analog.

Falls man auf die Unterstützung der Eingabedialoge verzichten möchte, kann man die Eingaben auch manuell in die entsprechenden Zellen eintragen. Allerdings muss in diesem Fall auch die Nummerierung der Knoten, Materialien bzw. Stäbe manuell und korrekt vorgenommen werden, da sonst der Einlese-Algorithmus nicht funktioniert!

### 4.5.2.3 Lastfall eingeben

Nach erfolgter Eingabe der Ausgangsdaten, nämlich den Systemknoten, den Materialien mit ihren Querschnitten und den System–Stäben können nun durch Klicken auf den Button „Lastfall eingeben“ die Lastfälle eingegeben werden. Lastfall 1 ist ein besonderer Lastfall, denn er muss die ständigen Lasten enthalten. An diesen Umstand wird durch eine Meldung erinnert.




Bild 140. Eingabe von Lastfall 1

Nach Bestätigung dieser Meldung werden wiederum die alten Eingaben gelöscht und es erscheint analog zu der in Kapitel 0 beschriebenen Eingabe eine Dialogreihe, die zur Eingabe der System–Kräfte auffordert. Nach abgeschlossener Kräfte–Eingabe kann in jedem Lastfall durch Klicken auf den Button „weiteren Lastfall eingeben“ ein weiterer Lastfall eingegeben werden. Für jeden Lastfall wird ein eigenes Arbeitsblatt mit entsprechender Benennung angelegt. Bei der Eingabe jedes weiteren Lastfalles muss die Lastfallnummer eingegeben werden. Es ist darauf zu achten, dass die Nummerierung der Lastfälle korrekt ist, da es sonst während des Einlese–Algorithmus zu Fehlern kommen kann! Es können maximal 50 Lastfälle eingegeben werden!

#### 4.5.2.4 Berechnung starten

Nachdem alle Lastfälle eingegeben wurden, muss zum Starten des Einlese- und Berechnungsalgorithmus der Button „Berechnung starten“ auf dem Arbeitsblatt „Tragwerk“ gedrückt werden. Als erstes wird eine manuelle Bestätigung der Anzahl der eingegebenen Knoten, Stäbe und Lastfälle abgefragt, die gegebenenfalls korrigiert werden muss. Diese Eingaben sind wichtig für den weiteren Verlauf der Berechnungen und sollten sorgfältig überprüft werden, da es sonst zu Fehlern kommen kann!

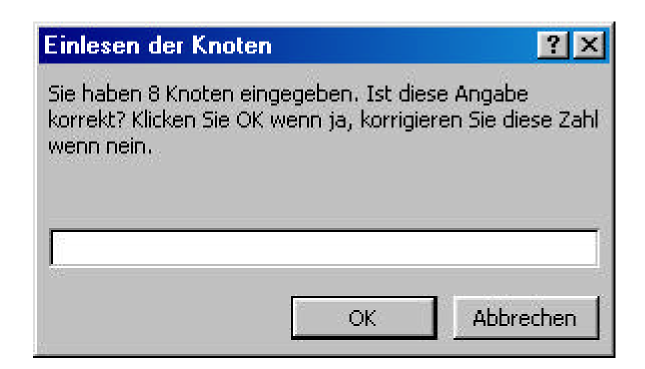



Bild 141. Manuelle Bestätigung der Anzahl der Knoten

Nun werden die Daten eingelesen und die Ergebnisse für jeden Lastfall berechnet. Gleichzeitig werden sie in den dafür vorgesehen Bereichen des jeweiligen Arbeitsblattes (=Lastfall) ausgegeben. Folgende Größen werden für das spezifizierte Tragwerk berechnet: Knotenverschiebungen, Stab–Spannungen, Stab–Kräfte.

Zum Schluss werden die Ergebnisse der einzelnen Lastfälle einer MIN/MAX – Überlagerung unterzogen. Das Ergebnis der Überlagerung wird auf dem Arbeitsblatt „Tragwerk“ unter „Ergebnisse der Lastfallüberlagerung“ angezeigt.

### 4.5.3 Verwendung in Mathcad[4]

Der Aufruf der Funktionen wurde bereits in Kapitel II-1.6.2 kurz erläutert. Er erfolgt in Mathcad über das Dialogfensters "Funktion einfügen", siehe Bild 143 .

[4] Verwendet wurde hier die Version Mathcad 2000. Wenn im Folgenden nur noch von Mathcad gesprochen wird, ist hierbei dieses Programm gemeint

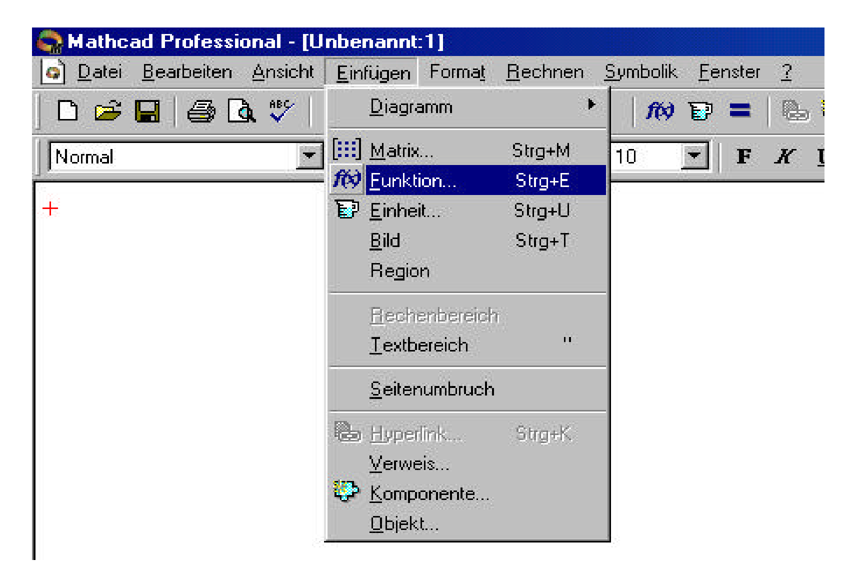



Bild 142. Menü Einfügen der Mathcad Anwendung

Der Aufruf des Dialogfensters geschieht über die Menüleiste in Mathcad (Bild 142). Die wesentlichen Informationen zu den Funktionen, welche auch in diesem Abschnitt der Dokumentation enthalten sind, wie Eingabeparameter oder auch Rückgabewerte, sind im unteren Teil des Dialogfensters als „Onlinehilfe“ verfügbar.

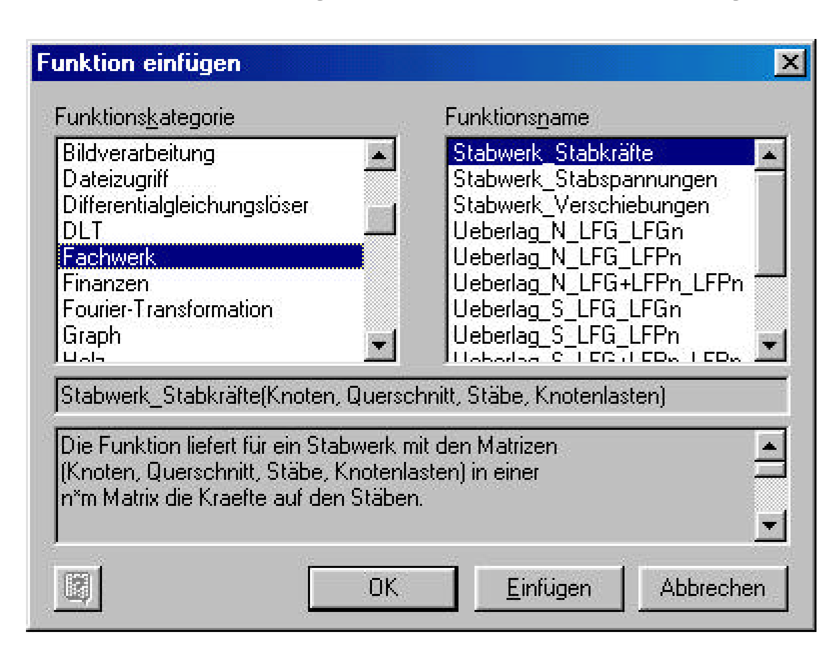



Bild 143. Dialogfenster Funktion einfügen in Mathcad 2000

Die implementierte Funktionalität, also die Berechnung von horizontalen und vertikalen Verschiebungen in einem Tragwerk, sowie die Ermittlung von Kräften und Spannungen auf den Stäben, wird von insgesamt 12 Funktionen wiedergegeben.

Diese Funktionen ermöglichen es, Lastfälle im einzelnen zu berechnen sowie diese Ergebnisse miteinander weiter zu überlagern. Dabei muss vom Anwender unterschieden werden, ob es sich bei dieser Überlagerung der Knotenverschiebungen und Kräfte in den Stäben sowie der Spannungen in den Stäben, um einen ersten und einen weiteren Lastfall oder um bereits überlagerte Lastfälle handelt. Für jede Unterscheidung ist eine spezielle Funktion

vorgesehen. Zudem ist eine „einfache“ Überlagerung möglich, die Maximal- und Minimal-Werte nicht unterscheidet und die Maximal- und Minimal-Ergebnisse überlagert.

Die Matrizen, die für die Berechnung der Lastfälle und der Überlagerungen der Funktionen übergeben werden, müssen im Aufbau exakt den Beschreibungen entsprechen. In Mathcad können Matrizen mit maximal 100 Elementen erstellt werden. Entsprechend ist die Knotenanzahl auf 20 beschränkt. Die Stabanzahl darf 25 nicht überschreiten. Die Funktionen werden im Folgenden noch einmal aufgeführt.

#### 4.5.3.1 Funktionen für die Berechnung von Lastfällen

Stabwerk_Verschiebungen

Stabwerk_Stabkräfte

Stabwerk_Stabspannungen

**Stabwerk_Verschiebungen**

Die Funktion liefert für ein Stabwerk mit den Matrizen (Knoten, Querschnitt, Stäbe, Knotenlasten) in einer n*m Matrix zeilenweise die Verschiebungen in x- und y-Richtung.

Übergabeparameter: Knoten, Querschnitt, Stäbe, Knotenlasten

| | |
|---|---|
| Knoten: | 1.Zeile: Knotennummer (1, 2, ...maximal 20)<br>2.Zeile: x-Koordinate<br>3.Zeile: y-Koordinate<br>4.Zeile: Auflager in x-Richtung: 1:ja - 0:nein<br>5.Zeile: Auflager in y-Richtung: 1:ja - 0:nein |
| Querschnitt: | 1.Zeile: Materialnummer (1, 2, ...)<br>2.Zeile: Fläche (in qm)<br>3.Zeile: e-Modul (in KN/qm)<br>4.Zeile: eine optionale Materialbenennung |
| Stäbe: | 1.Zeile: Stabnummer (1, 2, ...maximal 25)<br>2.Zeile: Anfangsknoten<br>3.Zeile: Endknoten<br>4.Zeile: Materialnummer |
| Knotenlasten: | 1.Zeile: Knotennummer (beliebige Reihenfolge)<br>2.Zeile: Kraft in x-Richtung<br>3.Zeile: Kraft in y-Richtung |

**Stabwerk_Stabkräfte**

Die Funktion liefert für ein Stabwerk mit den Matrizen (Knoten, Querschnitt, Stäbe, Knotenlasten) in einer n*m Matrix die Kraefte auf den Stäben.

Übergabeparameter: Knoten, Querschnitt, Stäbe, Knotenlasten

Knoten: 1.Zeile: Knotennummer (1, 2, ...maximal 20)
2.Zeile: x-Koordinate
3.Zeile: y-Koordinate
4.Zeile: Auflager in x-Richtung: 1:ja - 0:nein
5.Zeile: Auflager in y-Richtung: 1:ja - 0:nein

Querschnitt: 1.Zeile: Materialnummer (1, 2, ...)
2.Zeile: Fläche (in qm)
3.Zeile: e-Modul (in KN/qm)
4.Zeile: eine optionale Materialbenennung

Stäbe: 1.Zeile: Stabnummer (1, 2, ...maximal 25)
2.Zeile: Anfangsknoten
3.Zeile: Endknoten
4.Zeile: Materialnummer

Knotenlasten: 1.Zeile: Knotennummer (beliebige Reihenfolge)
2.Zeile: Kraft in x-Richtung
3.Zeile: Kraft in y-Richtung

**Stabwerk_Stabspannungen**

Die Funktion liefert für ein Stabwerk mit den Matrizen (Knoten, Querschnitt, Stäbe, Knotenlasten) in einer n*m Matrix die Spannungen auf den Stäben.

Übergabeparameter: Knoten, Querschnitt, Stäbe, Knotenlasten

Knoten: 1.Zeile: Knotennummer (1, 2, ...maximal 20)
2.Zeile: x-Koordinate
3.Zeile: y-Koordinate
4.Zeile: Auflager in x-Richtung: 1:ja - 0:nein
5.Zeile: Auflager in y-Richtung: 1:ja - 0:nein

Querschnitt: 1.Zeile: Materialnummer (1, 2, ...)
2.Zeile: Fläche (in $m^2$)
3.Zeile: E-Modul (in $KN/m^2$)
4.Zeile: Materialbenennung (optional)

Stäbe: 1.Zeile: Stabnummer (1, 2, ...maximal 25)
2.Zeile: Anfangsknoten
3.Zeile: Endknoten
4.Zeile: Materialnummer

Knotenlasten: 1.Zeile: Knotennummer (beliebige Reihenfolge)
2.Zeile: Kraft in x-Richtung
3.Zeile: Kraft in y-Richtung

#### 4.5.3.2 Funktionen für die Berechnung der jeweiligen Überlagerungen

Ueberlag_w_LFG_LFPn

Ueberlag_w_LFG+LFPn_LFPn

Ueberlag_w_LFG_LFGn

Ueberlag_N_LFG_LFPn

Ueberlag_N_LFG+LFPn_LFPn

Ueberlag_N_LFG_LFGn

Ueberlag_S_LFG_LFPn

Ueberlag_S_LFG+LFPn_LFPn

Ueberlag_S_LFG_LFGn

**Überlagerungen der Knotenverschiebung**

***Ueberlag_w_LFG_LFPn***

Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Verschiebung in einer n*m Matrix zeilenweise die MIN/MAX Werte der Knotenverschiebungen, Max_x, Min_x,Max_y,Min_ y.

Übergabeparameter: Lastfall1, Lastfall2

Eingabe:
1. Ergebnismatrix aus Lastfall ständige Last
2. Ergebnismatrix aus Lastfall Verkehr

***Ueberlag_w_LFG+LFPn_LFPn***

Die Funktion liefert aus der Ergebnismatrix der Funktion Ueberlag_w_LFG_LFPn und der Funktion Stabwerk_Verschiebung in einer n*m Matrix zeilenweise die MIN/MAX Werte der Knotenverschiebungen, Max_x, Min_x,Max_y,Min_ y.

Übergabeparameter: ErgebnisKnotenverschiebung1, Lastfall3

1. Ergebnismatrix aus Ueberlagerung (Ueberlag_w_LFG_LFPn)
2. Ergebnismatrix aus Lastfall Verkehr

***Ueberlag_w_LFG_LFGn***

Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Verschiebungen in einer n*m Matrix zeilenweise die MIN/MAX Werte der Knotenverschiebungen, Max_x, Min_x,Max_y,Min_ y.

Übergabeparameter: ErgebnisKnotenverschiebung1, Lastfall3

Eingabe:
1. Ergebnismatrix aus Lastfall ständige Last
2. Ergebnismatrix aus Lastfall ständige Last

**Überlagerung der Stabkräfte:**

**Ueberlag_N_LFG_LFPn**

Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Stabkräfte in einer n*m Matrix zeilenweise die MIN/MAX Werte der Stabkräfte, Max_N, Min_N.

Übergabeparameter: Stabkräfte1, Stabkräfte2

1. Ergebnismatrix aus Lastfall ständige Last
2. Ergebnismatrix aus Lastfall Verkehr

**Ueberlag_N_LFG+LFPn_LFPn**

Die Funktion liefert aus der Lösung aus Ueberlag_N_LFG_LFPn und den Ergebnismatrizen der Funktion Stabwerk_Stabkräfte in einer n*m Matrix zeilenweise die MIN/MAX Werte der Stabkräfte, Max_N, Min_N.

Übergabeparameter: Ergebnis1, Stabkräfte2

1. Ergebnismatrix aus der Überlagerung (Ueberlag_N_LFG_LFPn)
2. Ergebnismatrix aus Lastfall Verkehr

**Ueberlag_N_LFG_LFGn**

Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Stabkräfte in einer n*m Matrix zeilenweise die MIN/MAX Werte der Stabkräfte, Max_N, Min_N. (Überlagerung der ständigen Lasten)

Übergabeparameter: Ergebnis1, Stabkräfte2

1. Ergebnismatrix aus Lastfall ständige Last
2. Ergebnismatrix aus Lastfall ständige Last

**Überlagerung der Stabspannungen**

**Ueberlag_S_LFG_LFPn**

Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Stabspannungen in einer n*m Matrix zeilenweise die MIN/MAX Werte die Stabkräfte, Max_S, Min_S.

Übergabeparameter: Stabkräfte1, Stabkräfte2

1. Ergebnismatrix aus Lastfall ständige Last
2. Ergebnismatrix aus Lastfall Verkehr

**Ueberlag_S_LFG+LFPn_LFPn**

Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Stabspannungen und der Lösung aus Ueberlag_S_LFG_LFPn in einer n*m Matrix zeilenweise die MIN/MAX Werte der Stabkräfte, Max_S, Min_S.
Übergabeparameter: Ergebnis1, Stabkräfte2

1. Ergebnismatrix aus der Überlagerung (Ueberlag_S_LFG_LFPn)
2. Ergebnismatrix aus Lastfall Verkehr

**Ueberlag_S_LFG_LFGn**

Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Stabspannungen in einer n*m Matrix zeilenweise die Überlagerung der MIN/MAX Werte der Stabkräfte, Max_S, Min_S.

Übergabeparameter: Ergebnis1, Stabkräfte2

1. Ergebnismatrix aus Lastfall ständige Last
2. Ergebnismatrix aus Lastfall ständige Last

## 4.6 Ebenes Stabwerk

### 4.6.1 Allgemeines

Die Komponenten und Dateien zur Berechnung von ebenen Stabwerken wurde ebenfalls innerhalb eines Studentischen Projekts[5] erstellt.

### 4.6.2 Verwendung des Softwaremoduls in MS-Excel

Die Excel-Datei „Stabwerk.xls“ ist dem Erscheinungsbild der „Fachwerk.xls“ sehr ähnlich (Bild 144).

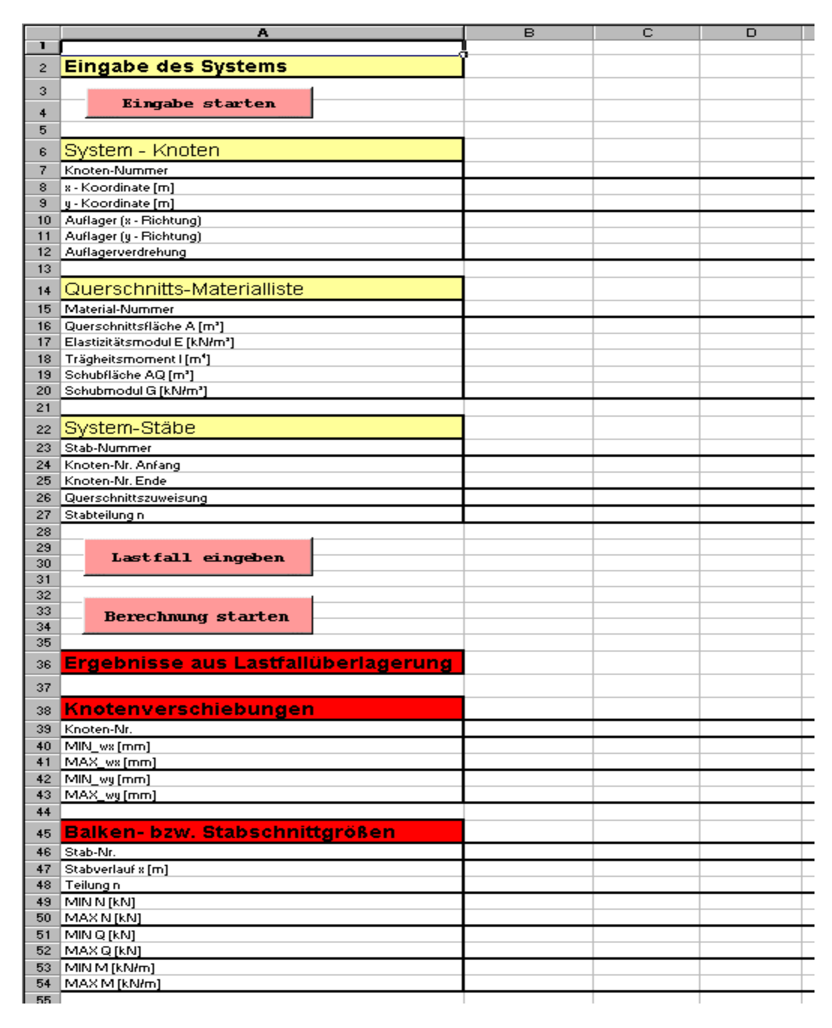



Bild 144. Startbildschirm für die Berechnung von ebenen Stabwerken in MS-Excel

Die Datei ist ebenfalls im Installationsordner des Softwarebaukastens zu finden. Auch ist die Bedienung nahezu identisch wie bei der Berechnung von Fachwerken. Somit kann hier auf die Kapitel 0 bis 4.5.2.4 verwiesen werden. Dies hat für den Anwender den Vorteil, dass die Einarbeitungszeiten minimal sind. Nach dem der Anwender auf den Button „Eingabe starten“ geklickt hat, informiert das Programm zunächst über die Koordinaten- und Kräftedefinition bevor es vom Anwender die System-Knoten abfragt, siehe Bild 145.

[5] Projekt EDATRA 2 im Fachbereich Wirtschaftsinformatik an der Fachhochschule Konstanz, Sommersemester 2001

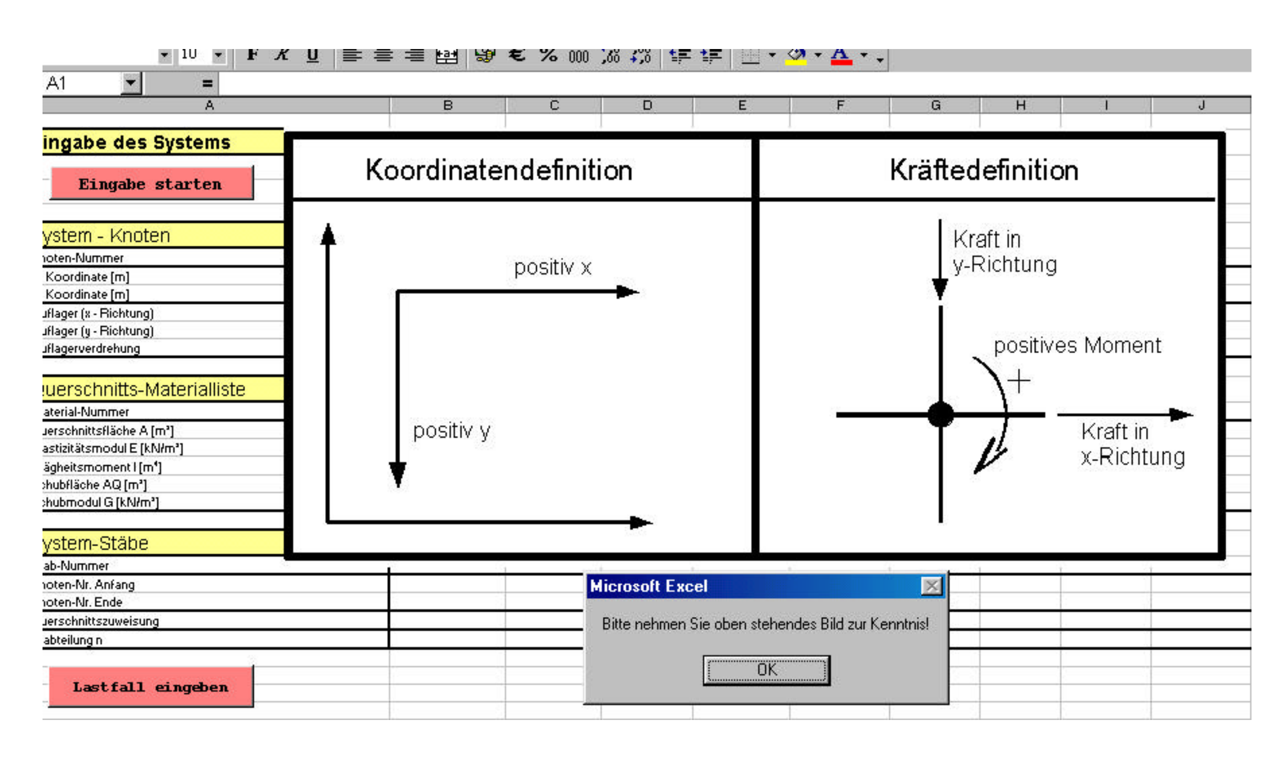



Bild 145. Definition der positiven Koordinaten- und Kraftrichtungen

Die Eingabe der Werte erfolgt über Input-Boxen[6], wie nachstehendes Bild 146 oder auch Bild 141 aus Kapitel 4.5.2.4 zeigt.

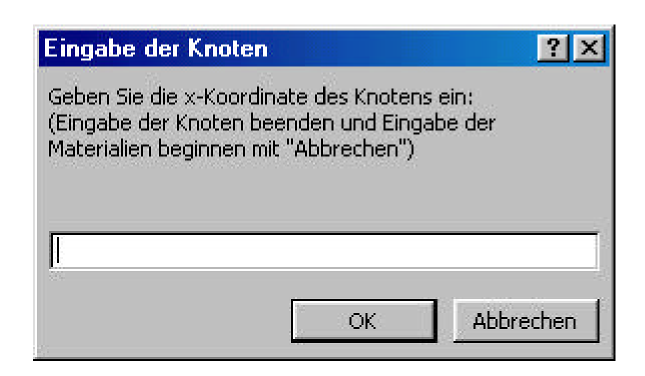



Bild 146. Input-Box für die Eingabe eines Knotens

Nachdem die Eingabe mit „Abbrechen“ oder der [Esc]-Taste beendet wurde, erwartet das Programm als nächste Eingabe der Querschnitts-Materialliste. Zuletzt müssen die System-Stäbe eingeben werden. Diese Systemdaten müssen jedoch nicht mit Hilfe der Input-Boxen eingegeben werden. Der Anwender kann die Daten auch direkt in das Excel-Arbeitsblatt schreiben. Es stehen somit zwei Eingabemöglichkeiten zur Verfügung.

Um die Daten für einen Lastfall zu erfassen, klickt der Anwender auf den Button „Lastfall eingeben“. Da im ersten Lastfall, wie bei der Berechnung von Fachwerken, die ständigen Lasten eingeben werden müssen, erscheint vor der Dateneingabe ein Hinweis auf diesen

[6] Input-Box – Steuerelement in VBA aus der Microsoft-Forms-Bibliothek

Umstand. Nachdem das Programm auf das Arbeitsblatt „Lastfall 1“ gewechselt hat, wird zuerst nach dem Lastfallfaktor gefragt, mit dem dieser Lastfall gewichtet werden soll. Der Lastfallfaktor wurde hauptsächlich wegen späterer Überlagerungen eingebaut, da nach dem Eurocode 1 mit seinem globalen Sicherheitskonzept, ständige Lasten und Verkehrslasten mit unterschiedlichen Faktoren berücksichtigt werden. Dieser Lastfallfaktor wird bereits im Excel-Makro berücksichtigt und die schon multiplizierten Lasten an das Berechnungsmodul „übergeben“.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Eingabe der Einwirkungen pro Lastfall | | | | |
| 2 | | | | | |
| 3 | Knoten-Lasten | | | | |
| 4 | Knoten-Nr. | | | | |
| 5 | Kraft in X [kN] | | | | |
| 6 | Kraft in Y [kN] | | | | |
| 7 | Stab-Lasten eingeben | | | | |
| 8 | Stab-Lasten | | | | |
| 9 | Stab-Nr. | | | | |
| 10 | Last am Stabanfang qA [kN/m] | | | | |
| 11 | Last am Stabende qB [kN/m] | | | | |
| 12 | Temperatur-Lasten eingeben | | | | |
| 13 | Temperatur-Lasten | | | | |
| 14 | Stab-Nr. | | | | |
| 15 | Temperaturdifferenz ΔT [°C] | | | | |
| 16 | Temperaturkoeffizient αT [1/K] | | | | |
| 17 | Balkenhöhe h [m] | | | | |
| 18 | | | | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | weiteren Lastfall eingeben | | | | |
| 22 | | | | | |
| 23 | | | | | |
| 24 | | | | | |
| 25 | Ergebnisse pro Lastfall | | | | |
| 26 | | | | | |
| 27 | Knotenverschiebungen | | | | |
| 28 | Knoten-Nr. | | | | |
| 29 | wx [mm] | | | | |
| 30 | wy [mm] | | | | |
| 31 | | | | | |
| 32 | | | | | |
| 33 | Balken- bzw. Stabschnittgrößen | | | | |
| 34 | Stab-Nr. | | | | |
| 35 | x [m] | | | | |
| 36 | Teilung | | | | |
| 37 | N [kN] | | | | |
| 38 | Q [kN] | | | | |
| 39 | M [kN/m] | | | | |
| 40 | | | | | |

Tragwerk | Lastfall 1

Bereit

Bild 147. Tabellenblatt für den Lastfall 1 bei der Berechnung von ebenen Stabwerken

Nachdem der Anwender die Kräfte auf die einzelnen Knoten eingeben hat, kann er über die Buttons „Stab-Lasten eingeben“, „Temperatur-Lasten eingeben“ oder „weiteren Lastfall eingeben“ die zugehörigen Daten eingeben oder einen neuen Lastfall anlegen. Wie auch auf dem ersten Arbeitsblatt kann der Anwender die Daten auch direkt (ohne Input-Boxen) eingeben.

Es ist zu beachten, dass maximal 50 Lastfälle eingegeben werde können. Die Lastfälle müssen in aufsteigender Reihenfolge erfasst werden und auch so bleiben. Ansonsten können Störungen bei der Berechnung auftreten.

Sind schließlich alle Lastfälle eingegeben worden, startet der Anwender mit einem Klick auf die Schaltfläche „Berechnung starten“ auf dem Arbeitsblatt „Tragwerk“ die Berechnung des Stabwerkes. Die berechneten Ergebnisse werden je Lastfall in den dafür vorgesehenen Zellen ausgegeben. Ebenso wird das Ergebnis der Lastfallüberlagerung auf dem Arbeitsblatt „Tragwerk“ ausgeben.

### 4.6.3 Verwendung des Softwaremoduls in Mathcad

Aufgrund des Umstandes, dass der Quellcode für die Java-Dll des Programms zur Berechnung ebener Stabwerke allein ca. 23.000 Zeilen Code umfasst, was etwa 600 Seiten ausmacht, war es aus Zeitgründen bis zur Erstellung des Abschlussberichts nicht möglich auch die Anbindung der Funktionalität an Mathcad abzuschließen, was jedoch innerhalb des oben erwähnten studentischen Projekts im Herbst dieses Jahres geschehen soll.

## 4.7 Stahlbetonbau

### 4.7.1 Allgemeines

Im Bereich des Stahlbetonbaus wurden mehrere Funktionen zur Biege- und Schubbemessung von Rechteckquerschnitten für Mathcad und MS-Excel entwickelt. Diese werden im Folgenden erläutert.

Da diese Berechnungen keine umfangreichen Eingaben erfordert, wurde lediglich für das $k_d$-Verfahren[7] nach DIN 1045-1 eine kleine Oberfläche programmiert. Für die restlichen Funktionen wurde in MS-Excel die Standardeingabe verwendet (siehe hierzu auch Kapitel II-1.4.1.1).

Für Mathcad ergibt sich wieder die in Kapitel 4.5.3 bereits erläuterte Vorgehensweise bei der Eingabe. Über die Menüleiste wird das Dialogfenster *Funktion einfügen* aufgerufen und die gewünschte Funktion ausgewählt. Die Funktionen sind in der Kategorie *Stahlbetonbau* zu finden Hierbei ergibt sich wieder der Umstand, dass in Mathcad keine Zeichenfolgen eingegeben werden können.

### 4.7.2 Biegebemessung

Folgende Tabelle zeigt die programmierten Funktionen für die Biegebemessung im Stahlbetonbau für MS-Excel und Mathcad.

| Funktionen in MS-Excel | Funktionen in Mathcad |
|---|---|
| Biegebem_Normalb_BSt500<br>Biegebem_Normalb_BSt500_Druckseite<br>Biegebem_Normalb_BSt500_Zugseite | BBem |
| Biegebemessung nach dem<br>$k_h$-Verfahren (DIN 1045) | |

Tabelle 16. Stahlbetonbau Funktionen für die Biegebemessung in MS-Excel und Mathcad

Die Ergebnisse einer Biegebemessung enthält im Allgemeinen Werte für die einzulegende Stahlmenge im Zugbereich und falls erforderlich auch die des im Druckbereichs. Da die Ausgabe von zwei Werten in Mathcad kein Problem darstellt, war hier lediglich eine Funktion nötig (*BBem*).

In MS-Excel hingegen verhält es sich bei der Standardmöglichkeit der Eingabe so, dass nur ein Wert in die markierte Zelle ausgegeben werden kann, auf andere Zellen besteht in solchen

[7] kh-Verfahren. Bemessungsverfahren nach DIN 1045

Funktionen keine Zugriffsmöglichkeit. Um diesem Umstand zu berücksichtigen, bestehen mehrere Möglichkeiten dennoch zwei Ergebnisse in das Tabellenblatt auszugeben.

Eine Möglichkeit besteht darin, beide Ergebnisse in einem String auszugeben. Allerdings kann dann in anderen Zellen des Arbeitsblattes in Formeln nicht mehr auf die Ergebniszellen der Berechnung verwiesen werden kann, da diese nicht als Zahlwerte vorliegen. Diese Vorgehensweise wurde bei der Funktion *Biegebem_Normalb_BSt500* gewählt.

Weiterhin besteht die Möglichkeit, mehrere Funktionen zu implementieren, von denen eine nur das Ergebnis für den Zugbereich (*Biegebem_Normalb_BSt500_Zugseite*) liefert und einen andere das Ergebnis für den Druckbereich (*Biegebem_Normalb_BSt500_Druckseite*). Dies ermöglicht wieder Verweise auf diese Ergebniszellen in anderen Funktionen des Arbeitsblatts, da die Ergebnisse nun als Zahlwerte vorliegen. Es bleibt zu sagen, dass alle drei Funktionen jedoch auf die gleiche Methode derselben Java-Dll zugreifen.

Eine weitere Möglichkeit besteht darin, eine eigene Benutzeroberfläche für das Programm zu erstellen. Dies ermöglicht es, auf beliebige Zellen eines Arbeitsblatts zuzugreifen. Eine eigene Oberfläche wurde für das $k_d$-Verfahren[8] programmiert. Zudem den Vorteil bietet eine eigene Oberfläche den Vorteil, dass nicht unbedingt eine Onlinehilfe erforderlich ist, da die Oberfläche mit entsprechenden Hinweisen zur Eingabe der Werte, wie Einheiten und dergleichen, versehen werden kann.

**Biegebem_Normalb_BSt500_Zugseite**

Bemessung von Betonquerschnitten nach DIN 1045-1 mit dem dimensionslosen OMEGA-Verfahren. Die Bemessung beschränkt sich auf Rechteckquerschnitte. Es können Biegemomente und Normalkräfte Berücksichtigt werden.

Rückgabewert ist die erforderliche Biegebewehrung auf der Zugseite des Querschnitts in [cm²]. Die Ausgabe des Werts erfolgt als Zahlwert.

Folgende Eingabeparameter sind vorzugeben:

| | |
|---|---|
| Querschnittsbreite | [m] |
| Querschnittshöhe | [m] |
| Abstand des Querschnittsrandes zur Bewehrung (symmetrisch für Zug- und Druckbewehrung) | [m] |
| Bemessungsmoment | [kNm] |
| Bemessungsnormalkraft | [kN] |
| Bemessungs-Betondruckfestigkeit | [MN/m²] |

[8] Das kd-Verfahren ist eine weitere Möglichkeit der Biegebemessung um Stahlbetonbau nach DIN 1045-1

**Biegebem_Normalb_BSt500_Druckseite**

Bemessung von Betonquerschnitten nach DIN 1045-1 mit dem dimensionslosen OMEGA-Verfahren. Die Bemessung beschränkt sich auf Rechteckquerschnitte. Es können Biegemomente und Normalkräfte Berücksichtigt werden.

Rückgabewert ist die erforderliche Biegebewehrung auf der Druckseite des Querschnitts in [cm²]. Die Ausgabe des Werts erfolgt als Zahlwert.

Folgende Eingabeparameter sind vorzugeben:

| | |
|---|---|
| Querschnittsbreite | [m] |
| Querschnittshöhe | [m] |
| Abstand des Querschnittsrandes zur Bewehrung (symmetrisch für Zug- und Druckbewehrung) | [m] |
| Bemessungsmoment | [kNm] |
| Bemessungsnormalkraft | [kN] |
| Bemessungs-Betondruckfestigkeit | [MN / m²] |

**Biegebem_Normalb_BSt500**

Bemessung von Betonquerschnitten nach DIN 1045-1 mit dem dimensionslosen OMEGA-Verfahren. Die Bemessung beschränkt sich auf Rechteckquerschnitte. Es können Biegemomente und Normalkräfte berücksichtigt werden.

Rückgabewert ist die erforderliche Biegebewehrung auf der Zugseite und auf der Druckseite des Querschnitts in [cm²]. Die Ausgabe beider Werte erfolgt als String.

Folgende Eingabeparameter sind vorzugeben:

| | |
|---|---|
| Querschnittsbreite | [m] |
| Querschnittshöhe | [m] |
| Abstand des Querschnittsrandes zur Bewehrung (symmetrisch für Zug- und Druckbewehrung) | [m] |
| Bemessungsmoment | [kNm] |
| Bemessungsnormalkraft | [kN] |
| Bemessungs-Betondruckfestigkeit | [MN / m²] |

**BBem**

Diese Funktion ist lediglich in Mathcad verfügbar und ermöglicht die Bemessung von Betonquerschnitten nach DIN 1045-1 mit dem dimensionslosen OMEGA-Verfahren. Die Bemessung beschränkt sich auf Rechteckquerschnitte. Es können Biegemomente und Normalkräfte berücksichtigt werden.

Folgende Eingabeparameter sind vorzugeben:

Breite, Hoehe, e_Bewehrung, $M_{sd}$, $N_{sd}$, $f_{cd}$

Eingabe:
Querschnittsbreite - [m]
Querschnittshöhe - [m]
Abstand des Querschnittsrandes zur Bewehrung [m]
(symetrisch für Zug- und Druckbewehrung)

Bemessungsmoment - [kNm]

Bemessungsnormalkraft . [kN]

Bemessungs-Betondruckfestigkeit [MN / m^2]

**Bemessung nach dem $k_d$-Verfahren nach DIN 1045-1**

Bemessung von Betonquerschnitten nach DIN 1045-1 mit dem dimensionsgebundenen $k_d$-Verfahren. Die Bemessung beschränkt sich auf Rechteckquerschnitte, Betonstahl der Güte BSt 500 und Beton der Festigkeitsklasse $\leq$ C50/60. Es können Biegemomente und Normalkräfte berücksichtigt werden.

Rückgabewert ist die erforderliche Biegebewehrung auf der Zugseite in [cm²]. Die Ausgabe beider Werte erfolgt als Zahl. Für den Fall, dass Druckbewehrung erforderlich wird, erfolgt eine Hinweismeldung.

Folgende Eingabeparameter sind vorzugeben:

| | |
|---|---|
| Querschnittshöhe | [cm] |
| Querschnittsbreite | [cm] |
| Statische Höhe | [cm] |
| Betonfestigkeit C | [MN / m²] |
| Bemessungsmoment | [kNm] |
| Bemessungsnormalkraft | [kN] |

### 4.7.3 Schubbemessung

Folgende Tabelle zeigt die Funktionen für die Schubbemessung im Stahlbetonbau nach DIN 1045-1 für MS-Excel und Mathcad. Es wurden wieder mehrere Funktionen in Mathcad programmiert.

| Funktionen in MS-Excel | Funktionen in Mathcad |
|---|---|
| querkraftBemessung | NormalB_Bst500_schraeg<br>NormalB_Bst500_VertikaleBuegel<br>LeichtB_Bst500_schraeg<br>LeichtB_Bst500_VertikaleBuegel |

Tabelle 17. Stahlbetonbau Funktionen für die Schubbemessung in MS-Excel und Mathcad

**querkraftBemessung**

Die Funktion ist nur in MS-Excel verfügbar und stellt die Querkaftbemessung von Stahlbeton-Rechteckquerschnitten nach DIN 1045-1 für Leicht- und Normalbeton zur Verfügung. Die Funktion liefert den erforderlichen Gesamtquerschnitt der Schubbewehrung in [cm²] für senkrechte und unter dem Winkel alpha in [°] geneigte Bügelbewehrung der Betonstahlsorte BSt 500.

Die Eingabeparameter sind:

| | |
|---|---|
| Vsd_kN | Bemessungsquerkraft [kN] |
| Stat_hoehe_m | Kleinste Querschnittsbreite in [m] |
| Fck_MnproM² | Nennwert der Betondruckfestigkeit [MN / m^2] |
| Asl_cm² | Eingelegte Längsbewehrung [cm^2] |
| SigmCD_N_MM² | vorhandene Betonspannungen [MN / m^2] z.B. infolge Vorspannung, Zwang |
| Alpha_grad | Neigung der Schubbewehrung in [Grad] |
| Z_m | Hebelarm der inneren Kräfte i.A. 0.9 * d [m] |
| Roh_KGproM³ | Spezifisches Raumgewicht des Betons [kg/m^3] (Für Normalbeton kann der Wert 1 eingegeben werden, dies gilt jedoch nicht für Leichtbeton) |
| Gamma_c | Sicherheitsbeiwert des Betons (i.A. gamma_c=1,5 ) |
| Fyk_NproMM² | Nennwert der Betonstahl Zugfestigkeit [N / mm²] |
| Gamma_S | Sicherheitsbeiwert des Betonstahls (i.A. gamma_S=1,15) |

**NormalB_Bst500_VertikaleBuegel**

Querkaftbemessung von Stahlbetonrechteckquerschnitten nach DIN 1045-1. Die Funktion liefert den erforderlichen Gesamtquerschnitt an Schubbewehrung in [cm²] für Normalbeton mit senkrechter Bügelbewehrung der Betonstahlsorte BSt 500

Die Eingabeparameter sind:

V_sd, d, bw, f_ck, A_sl, sigma_cd, z

Hierin bedeuten:

| | | |
|---|---|---|
| V_sd | - | Bemessungsquerkraft [kN] |
| d | - | Querschnittshöhe [m] |
| bw | - | kleinste Querschnittsbreite in [m] |
| f_ck | - | Nennwert der Betondruckfestigkeit [MN / m^2] |
| A_sl | - | Eingelegte Längsbewehrung [cm^2] |
| sigma_cd | - | vorhandene Betonspannungen [MN / m^2] z.B. infolge Vorspannung, Zwang |
| z | - | Hebelarm der inneren Kräfte i.A. 0.9 * d [m] |

**NormalB_Bst500_schraeg**

Querkaftbemessung von Stahlbeton-Rechteckquerschnitten nach DIN 1045-1 Die Funktion liefert den erforderlichen Gesamtquerschnitt an Schubbewehrung in [cm²] für Normalbeton mit schrägen Bügelbewehrung der Betonstahlsorte BSt 500.

Die Eingabeparameter sind:
V_sd, d, bw, f_ck, A_sl, sigma_cd, alpha, z

Hierin bedeuten:

| | | |
|---|---|---|
| V_sd | - | Bemessungsquerkraft [kN] |
| d | - | Querschnittshöhe [m] |
| bw | - | kleinste Querschnittsbreite in [m] |
| f_ck | - | Nennwert der Betondruckfestigkeit [MN / m^2] |
| A_sl | - | Eingelegte Längsbewehrung [cm^2] |
| sigma_cd | - | vorhandene Betonspannungen [MN / m^ 2] z.B. infolge Vorspannung, Zwang |
| alpha | - | Neigung der Schubbewehrung [°] |
| z | - | Hebelarm der inneren Kräfte i.A. 0.9 * d [m] |

**LeichtB_Bst500_VertikaleBuegel**

Querkaftbemessung von Stahlbeton-Rechteckquerschnitten nach DIN 1045-1 Die Funktion liefert den erforderlichen Gesamtquerschnitt an Schubbewehrung in [cm²] für Leichtbetone mit Eingabe des Rohgewichts mit senkrechter Bügelbewehrung der Betonstahlsorte BSt 500.

Die Eingabeparameter sind:
V_sd, d, bw, f_ck, A_sl, sigma_cd, z, ro

Hierin bedeuten:

| | | |
|---|---|---|
| V_sd | - | Bemessungsquerkraft [kN] |
| d | - | Querschnittshöhe [m] |
| bw | - | kleinste Querschnittsbreite in [m] |
| f_ck | - | Nennwert der Betondruckfestigkeit [MN / m²] |
| A_sl | - | Eingelegte Längsbewehrung [cm²] |
| sigma_cd | - | vorhandene Betonspannungen [MN / m²] z.B. infolge Vorspannung, Zwang |
| z | - | Hebelarm der inneren Kräfte i.A. 0.9 * d [m] |
| Roh | - | spezifisches Raumgewicht des Beton [kg/$m^3$] |

**LeichtB_Bst500_schraeg**

Querkaftbemessung von Stahlbetonrechteckquerschnitten nach DIN 1045-1 Die Funktion liefert den erforderlichen Gesamtquerschnitt an Schubbewehrung in [cm²] für Leichtbetone des Rohgewichts mit schräger Bügelbewehrung der Betonstahlsorte BSt 500.

Die Eingabeparameter sind:
V_sd, d, bw, f_ck, A_sl, Sigma_sd, alpha, z, ro

Hierin bedeuten:

| | | |
|---|---|---|
| V_sd | - | Bemessungsquerkraft [kN] |
| d | - | Querschnittshöhe [m] |
| bw | - | kleinste Querschnittsbreite in [m] |
| f_ck | - | Nennwert der Betondruckfestigkeit [MN / m^2] |
| A_sl | - | Eingelegte Längsbewehrung [cm^2] |
| sigma_cd | - | vorhandene Betonspannungen [MN / m^2] z.B. infolge Vorspannung, Zwang |
| alpha | - | Neigung der Schubbewehrung [°] |
| z | - | Hebelarm der inneren Kräfte i.A. 0.9 * d [m] |
| Roh | - | spezifisches Raumgewicht des Beton [kg/$m^3$] |

## 4.8 Stahlbau

### 4.8.1 Allgemeines

Für Nachweise im Stahlbau wurden 40 Funktionen für Mathcad und 16 Funktionen für MS-Excel entwickelt. Diese werden nachfolgend erläutert.

Die hier entwickelten Funktionen erfordern keine umfangreicheren Eingaben. Somit war für Excel keine eigene Oberfläche nötig und es konnte auf die Standardmöglichkeit der Eingabe zurückgegriffen werden. Siehe hierzu auch Kapitel II-1.4.1.1.

Für Mathcad ergibt sich wieder die in Kapitel 4.5.3 bereits erläuterte Vorgehensweise bei der Eingabe. Über die Menüleiste wird das Dialogfenster *Funktion einfügen* aufgerufen und die gewünschte Funktion ausgewählt. Die Funktionen sind in der Kategorie *Stahlbau* zu finden Hierbei ergibt sich wieder der Umstand, dass in Mathcad keine Zeichenfolgen eingegeben werden können, was die große Anzahl der erstellten Funktionen erklärt. Es wurde für jede Profilreihe, falls erforderlich, eine eigene Funktion erstellt.

### 4.8.2 Normalkraftbeanspruchung

| Funktionen in MS-Excel | Funktionen in Mathcad |
|---|---|
| StahlZug_Nachweis | StahlZug_Nachweis_I<br>StahlZug_Nachweis_IPE<br>StahlZug_Nachweis_HEA<br>StahlZug_Nachweis_HEB<br>StahlZug_Nachweis_HEM |
| StahlZug_Nachweis_A | StahlZug_Nachweis_A |
| StahlZug_zulN | StahlZug_zulN_I<br>StahlZug_zulN_IPE<br>StahlZug_zulN_HEA<br>StahlZug_zulN_HEB<br>StahlZug_zulN_HEM |
| StahlZug_zulN_A | StahlZug_zulN_A |
| StahlZug_Bem | StahlZug_Bem_I<br>StahlZug_Bem_IPE<br>StahlZug_Bem_HEA<br>StahlZug_Bem_HEB<br>StahlZug_Bem_HEM |
| StahlZug_Bem_A | StahlZug_Bem_A |
| StahlDruck_Nachweis | StahlDruck_Nachweis_I<br>StahlDruck_Nachweis_IPE<br>StahlDruck_Nachweis_HEA<br>StahlDruck_Nachweis_HEB<br>StahlDruck_Nachweis_HEM |

| Funktionen in MS-Excel | Funktionen in Mathcad |
|---|---|
| StahlDruck_zulN | StahlDruck_zulN_I<br>StahlDruck_zulN_IPE<br>StahlDruck_zulN_HEA<br>StahlDruck_zulN_HEB<br>StahlDruck_zulN_HEM |
| StahlDruck_Bem | StahlDruck_Bem _I<br>StahlDruck_Bem _IPE<br>StahlDruck_Bem _HEA<br>StahlDruck_Bem _HEB<br>StahlDruck_Bem _HEM |

Tabelle 18. Stahlbaufunktionen in MS-Excel und Mathcad

**StahlZug_Nachweis**

Excel:
Die Funktion liefert den Nachweis eines Zugstabes der Profilreihen I, IPE, HEA, HEB, HEM und für die Stahlgüte St37, St52

Eingabeparameter: N[kN], Profilbezeichnung, Stahlgüte

Eingabe: N - Normalkraft [kN]
Profilbezeichnung [z.B. I200]
Stahlgüte ["St37" bzw. "St52"]

Mathcad:
Hier wurden für die fünf unterschiedlichen Profile je eine Funktion erstellt. Der Funktionsname in Excel wurde dafür mit der Endung *_Profilbezeichnung* erweitert. Also beispielsweise *StahlZug_Nachweis_HEA.*

Eingabeparameter: N[kN], Profilnennhöhe[mm], Stahlgüte

Eingabe:
N - Normalkraft [kN]
Profilnennhöhe [mm]
Stahlgüte [37 / 52]

Angabe der Profilnennhöhe nach folgendem Muster:
*I450* wird eingegeben als *450*
Angabe der Stahlgüte nach folgendem Muster:
*St52*, bzw. *S355* wird eingegeben als *52*
*St37*, bzw. *S235* wird eingegeben als *37*

**StahlZug_Nachweis_A**

***Excel und Mathcad***
Die Funktion liefert den Nachweis eines Zugstabes aufgrund der Querschnittsfläche für die Stahlgüte St37, St52

Eingabeparameter: N[kN], A[cm²], Stahlgüte

Eingabe: N - Normalkraft [kN]
Querschnittsfläche A [cm²]

Stahlgüte [“St37” bzw. “St52”] (für Excel)
Stahlgüte [37 bzw. 52] (für Mathcad)

**StahlZug_zulN**

***Excel:***
Die Funktion liefert die zul. Normalkraft[kN] eines Zugstabes aufgrund der Profilreihen I, IPE, HEA, HEB, HEM für die Stahlgüte St37, St52

Eingabeparameter: Profilbezeichnung, Stahlgüte

Eingabe: Profilbezeichnung [z.B. I200]
Stahlgüte [“St37” bzw. “St52”]

***Mathcad:***
Hier wurden für die 5 unterschiedlichen Profile je eine Funktion erstellt. Der Funktionsname in Excel wurde dafür mit der Endung *_Profilbezeichnung* erweitert, also beispielsweise *StahlZug_zulN_HEA*.

Eingabeparameter: Profilnennhöhe[mm], Stahlgüte

Eingabe:
Profilnennhöhe [mm]
Stahlgüte [37 / 52]

Angabe der Profilnennhöhe nach folgendem Muster:
*I450* wird eingegeben als *450*
Angabe der Stahlgüte nach folgendem Muster:
*St52*, bzw. *S355* wird eingegeben als *52*
*St37*, bzw. *S235* wird eingegeben als *37*

**StahlZug_zulN_A**

Excel und Mathcad:
Die Funktion liefert die zul. Normalkraft[kN] eines Zugstabes aufgrund der Querschnittsfläche für die Stahlgüte St37, St52

Eingabeparameter: A[cm²], Stahlgüte

Eingabe: Querschnittsfläche A [cm²]
Stahlgüte [“St37” bzw. “St52”] (für Excel)
Stahlgüte [37 bzw. 52] (für Mathcad)

**StahlZug_Bem**

Excel:
Die Funktion liefert die erf. Profilnennhöhe[mm] eines Zugstabes aufgrund der Profilreihen I, IPE, HEA, HEB, HEM für die Stahlgüte St37, St52

Eingabeparameter: N[kN],Profilreihe, Stahlgüte

Eingabe: N - Normalkraft [kN]
Profilreihe [z.B. „HEA“]
Stahlgüte [“St37” bzw. “St52”]

Mathcad:
Hier wurden für die 5 unterschiedlichen Profile je eine Funktion erstellt. Der Funktionsname in Excel wurde dafür mit der Endung *_Profilbezeichnung* erweitert, also beispielsweise *StahlZug_Bem_HEA*.

Eingabeparameter: N[kN], Profilnennhöhe[mm], Stahlgüte

Eingabe:
N - Normalkraft [kN]
Profilnennhöhe [mm]
Stahlgüte [37 / 52]

Angabe der Profilnennhöhe nach folgendem Muster:
*I450* wird eingegeben als *450*
Angabe der Stahlgüte nach folgendem Muster:
*St52*, bzw. *S355* wird eingegeben als *52*
*St37*, bzw. *S235* wird eingegeben als *37*

**StahlZug_Bem_A**

Excel und Mathcad:
Die Funktion liefert den erf. Querschnitt[$cm^2$] eines Zugstabes für die Stahlgüte St37, St52

Eingabeparameter: N[kN], Stahlgüte

Eingabe:
N - Normalkraft [kN]

Stahlgüte [“St37” bzw. “St52”] (für Excel)
Stahlgüte [37 bzw. 52] (für Mathcad)

**StahlDruck_Nachweis**

Excel:
Die Funktion liefert den Nachweis eines Druckstabes der Profilreihen I, IPE, HEA, HEB, HEM für die Stahlgüte St37, St52

Eingabeparameter: N[kN], Profilbezeichnung, Stahlgüte, sky[cm],skz[cm]

Eingabe:
N - Normalkraft [kN]
Profilbezeichnung [z.B. „I200“]
Stahlgüte [“St37” bzw. “St52”]
Knicklänge sky [cm]
Knicklänge skz [cm]

Mathcad:
Hier wurden für die 5 unterschiedlichen Profile je eine Funktion erstellt. Der Funktionsname in Excel wurde dafür mit der Endung *Profilbezeichnung erweitert. Also beispielsweise StahlDruck_Nachweis_HEA.*

Eingabeparameter: N[kN], Profilnennhöhe[mm], Stahlgüte, sky[cm],skz[cm]

Eingabe: N - Normalkraft [kN]
Profilnennhöhe [mm]
Stahlgüte [37 bzw. 52]
Knicklänge sky [cm]
Knicklänge skz [cm]

Angabe der Profilnennhöhe nach folgendem Muster:
*I450* wird eingegeben als *450*
Angabe der Stahlgüte nach folgendem Muster:
*St52*, bzw. *S355* wird eingegeben als *52*
*St37*, bzw. *S235* wird eingegeben als *37*

**StahlDruck_zulN**

Excel:
Die Funktion liefert die zul. Normalkraft[kN] eines Druckstabes der Profilreihen I, IPE, HEA, HEB, HEM für die Stahlgüte St37, St52

Eingabeparameter: Profilbezeichnung, Stahlgüte, sky[cm],skz[cm]

Eingabe: Profilbezeichnung [z.B. „I200“]
Stahlgüte [“St37” bzw. “St52”]
Knicklänge sky [cm]
Knicklänge skz [cm]

Mathcad:
Hier wurden für die 5 unterschiedlichen Profile je eine Funktion erstellt. Der Funktionsname in Excel wurde dafür mit der Endung *Profilbezeichnung erweitert. Also beispielsweise StahlDruck_zulN_HEA.*

Eingabeparameter: Profilnennhöhe[mm], Stahlgüte, sky[cm],skz[cm]

Eingabe:
Profilnennhöhe [mm]
Stahlgüte [37 bzw. 52]
Knicklänge sky [cm]
Knicklänge skz [cm]

Angabe der Profilnennhöhe nach folgendem Muster:
*I450* wird eingegeben als *450*
Angabe der Stahlgüte nach folgendem Muster:
*St52*, bzw. *S355* wird eingegeben als *52*
*St37*, bzw. *S235* wird eingegeben als *37*

**StahlDruck_Bem**

Excel:
Die Funktion liefert die erforderlichen Profilnennhöhe [mm] eines Druckstabes der Profilreihen I, IPE, HEA, HEB, HEM für die Stahlgüte St37, St52
Eingabeparameter: N[kN], sky[cm],skz[cm], Stahlgüte, Profilreihe

Eingabe: N - Normalkraft [kN]
Knicklänge sky [cm]
Knicklänge skz [cm]
Stahlgüte [“St37” bzw. “St52”]
Profilreihe [z.B. “HEA“]

Mathcad:
Hier wurden für die 5 unterschiedlichen Profile je eine Funktion erstellt. Der Funktionsname in Excel wurde dafür mit der Endung *Profilbezeichnung erweitert. Also beispielsweise StahlDruck_Bem_HEA.*

Eingabeparameter: N[kN], sky[cm],skz[cm], Stahlgüte

Eingabe:
N - Normalkraft [kN]
Knicklänge sky [cm]
Knicklänge skz [cm]
Stahlgüte [37 bzw. 52]

Angabe der Stahlgüte nach folgendem Muster:
*St52*, bzw. *S355* wird eingegeben als *52*
*St37*, bzw. *S235* wird eingegeben als *37*

### 4.8.3 Biegung

In Tabelle 19 sind die Funktionen für den allgemeinen Spannungsnachweis in Excel und Mathcad aufgeführt. Die Programmierung der Funktionen erfolgte nach den Formeln aus den Schneider Bautabellen [1]. Grundlage hierfür ist DIN 18 800 (11.90). Die Unterschiede bei der Eingabe der Parameter in MS-Excel und Mathcad sind bei den Funktionen für den

allgemeinen Spannungsnachweis nur geringfügig. Daher wird in den einzelnen Funktions-beschreibungen nur auf die Unterschiede hingewiesen.

| Funktionen in MS-Excel | Funktionen in Mathcad |
|---|---|
| Sigma_Nachweis_My | Sigma_Nachweis_My |
| Sigma_Nachweis_Mz | Sigma_Nachweis_Mz |
| Sigma_Nachweis_zweiachsig | Sigma_Nachweis_zweiachsig |

Tabelle 19. Nachweisfunktionen in MS-Excel und Mathcad für den allgemeinen Spannungsnachweis

**Sigma_Nachweis_My**

Die Funktion führt den allgemeinen Spannungsnachweis für einachsige Biegung um die Y-Achse eines doppeltsymmetrischen I-Profils mit Normalkraft.

Rückgabewert für den Spannungsnachweis ist der Verhältniswert $S_d/R_d$.

Eingabe :

$N_d$, $M_{yd}$, $V_{zd}$, Profil, Stahlguete, Begrenzung

Eingabeparameter für Excel:

Einwirkung : - $N_d$ Normalkraft in [kN] (Druck mit negativem Vorzeichen)
- $M_{yd}$ Biegemoment um die Y-Achse in [kNm]
- $V_{zd}$ Querkraft in Z-Richtung in [kN]

Profil: z. B. [„I300“] (als “String”)

Stahlgüte ["St37” bzw. “S235”] (als “String”)

Begrenzung:
Für Einfeld- und Durchlaufträger müssen die plastischen Momente auf den 1.25-fachen Wert des Grenzbiegemomentes im elastischen Zustand reduziert werden. Bei Eingabe einer 1 wird die Begrenzung berücksichtigt. Bei Eingabe einer 0 wird die Begrenzung nicht berücksichtigt.

Eingabeparameter für Mathcad:

Einwirkung : - $N_d$ Normalkraft in [kN] (Druck mit negativem Vorzeichen)

- $M_{yd}$ Biegemoment um die Y-Achse in [kNm]

- $V_{zd}$ Querkraft in Z-Richtung in [kN]

Profil: [Profilkenn ung Profilhoeh e] 1*2 Matrix

| | |
|---|---|
| Profilkennung: | 1 - I-Profil<br>2 - IPE-Profil<br>3 - HEA-Profil<br>4 - HEB-Profil<br>5 - HEM-Profil |
| | Profilhöhe in [mm] |
| Stahlguete: | 1 - S235<br>2 - S355<br>3 - St37<br>4 - St52 |

Begrenzung: 1 oder 0. Siehe hierzu die Angaben unter *Eingabeparameter für Excel.*

**Sigma_Nachweis_Mz**

Die Funktion führt den allgemeinen Spannungsnachweis für einachsige Biegung um die Z-Achse mit Normalkraft.

Rückgabewert für den Spannungsnachweis ist der Verhältniswert $S_d/R_d$.

Eingabe :

$N_d$, $M_{zd}$, $V_{yd}$, Profil, Stahlgüte, Begrenzung

Eingabeparameter für Excel:

| | | |
|---|---|---|
| Einwirkung : | - $N_d$ | Normalkraft in [kN]<br>(Druck mit negativem Vorzeichen) |
| | - $M_{zd}$ | Biegemoment um die Z-Achse in [kNm] |
| | - $V_{yd}$ | Querkraft in Y-Richtung in [kN] |
| Profil: | z. B. [„I300“] | (als “String”) |
| Stahlgüte | [“St37” bzw. “S235”] | (als “String”) |

Begrenzung:
Für Einfeld- und Durchlaufträger müssen die plastischen Momente auf den 1.25-fachen Wert des Grenzbiegemomentes im elastischen Zustand reduziert werden. Bei Eingabe einer 1 wird die Begrenzung berücksichtigt. Bei Eingabe einer 0 wird die Begrenzung nicht berücksichtigt.

Eingabeparameter für Mathcad:

| | | |
|---|---|---|
| Einwirkung : | - $N_d$ | Normalkraft in [kN]<br>(Druck mit negativem Vorzeichen) |
| | - $M_{zd}$ | Biegemoment um die Z-Achse in [kNm] |

- $V_{yd}$ Querkraft in Y-Richtung in [kN]

Profil: [Profilkennung Profilhoehe] 1*2 Matrix

Profilkennung:
1 - I-Profil
2 - IPE-Profil
3 - HEA-Profil
4 - HEB-Profil
5 - HEM-Profil

Profilhöhe in [mm]

Stahlguete:
1 - S235
2 - S355
3 - St37
4 - St52

Begrenzung: 1 oder 0. Siehe hierzu die Angaben unter *Eingabeparameter für Excel.*

**Sigma_Nachweis_zweiachsig**

Die Funktion führt den allgemeinen Spannungsnachweis für zweiachsige Biegung mit Normalkraft.

Rückgabewert für den Spannungsnachweis ist der Verhältniswert $S_d/R_d$.

Eingabe:

Nd, Myd, Mzd, Vzd, Vyd, Profil, Stahlguete, Begrenzung

Einwirkung :
- $N_d$ Normalkraft in [kN] (Druck mit negativem Vorzeichen)
- $M_{yd}$ Biegemoment um die Y-Achse in [kNm]
- $M_{zd}$ Biegemoment um die Z-Achse in [kNm]
- $V_{zd}$ Querkraft in Z-Richtung in [kN]
- $V_{yd}$ Querkraft in Y-Richtung in [kN]

Profil: z. B. [„I300“] (als “String”)

Stahlgüte ["St37” bzw. “S235”] (als “String”)

Begrenzung:
Für Einfeld- und Durchlaufträger müssen die plastischen Momente auf den 1.25-fachen Wert des Grenzbiegemomentes im elastischen Zustand reduziert werden. Bei Eingabe einer 1 wird die Begrenzung berücksichtigt. Bei Eingabe einer 0 wird die Begrenzung nicht berücksichtigt.

Eingabeparameter für Mathcad:

Einwirkung : - $N_d$ Normalkraft in [kN]

(Druck mit negativem Vorzeichen)

- $M_{yd}$ Biegemoment um die Y-Achse in [kNm]
- $M_{zd}$ Biegemoment um die Z-Achse in [kNm]
- $V_{zd}$ Querkraft in Z-Richtung in [kN]
- $V_{yd}$ Querkraft in Y-Richtung in [kN]

Profil: $[Profilkennung \quad Profilhoehe]$ 1*2 Matrix

Profilkennung:
1 - I-Profil
2 - IPE-Profil
3 - HEA-Profil
4 - HEB-Profil
5 - HEM-Profil

Profilhöhe in [mm]

Stahlguete:
1 - S235
2 - S355
3 - St37
4 - St52

Begrenzung: 1 oder 0. Siehe hierzu die Angaben unter *Eingabeparameter für Excel.*

### 4.8.4 Biegedrillknicken

In Tabelle 20 sind die Funktionen für den Biegedrillknicknachweis in Excel und Mathcad aufgeführt.

| Funktionen in MS-Excel | Funktionen in Mathcad |
|---|---|
| Bdk_Nachweis_Zweiachsig | biegungZweiachsig |
| Bdk_Zweiachsig_MethodeII | biegungZweiachsigMethodeII |
| Bdk_Nachweis_einachsig_mitN | biegungEinachsig |
| Bdk_Nachweis_einachsig_ohneN | biegungEinachsigOhneN |

Tabelle 20. Nachweisfunktionen in MS-Excel und Mathcad für den Biegedrillknicknachweis

In Mathcad erscheinen die vier Funktionen in einer eigenen Kategorie mit dem Namen *BDK* , wie Bild 148 zeigt. So werden auch Verwechslungen mit anderen Nachweisfunktionen für den Stahlbau vermieden. In den folgenden Erläuterungen zu den Funktionen wird die Eingabe in MS-Excel und Mathcad gesondert behandelt. Die Unterschiede ergebne sich dadurch, dass

in Mathcad keine Strings übergeben werden können und maximal zehn Parameter an eine Funktion weitergegeben werden können.

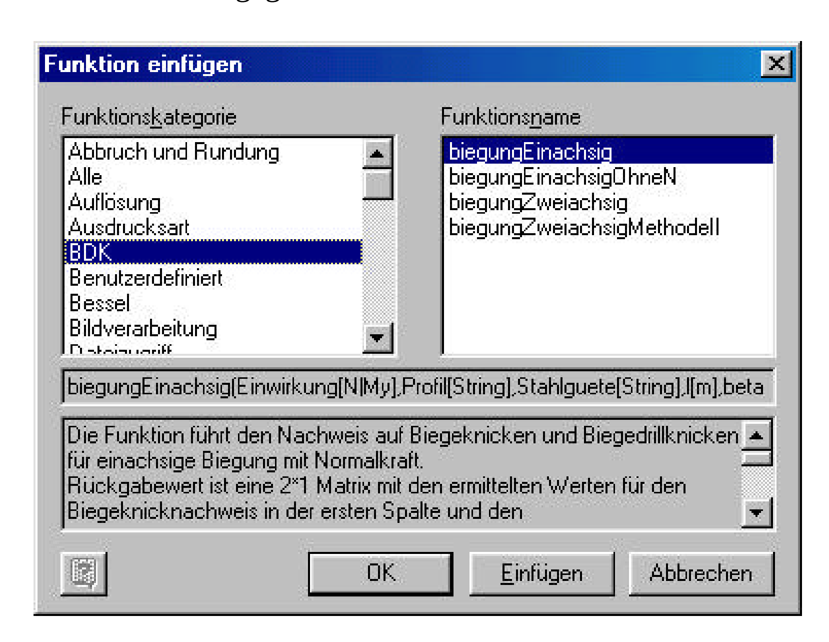



Bild 148. Funktionen für den Biegedrillknicknachweis in Mathcad

### 4.8.4.1 Nachweisfunktion für den Biegedrillknicknachweis in MS-Excel

**Bdk_Nachweis_Zweiachsig**

Die Funktion führt den Nachweis auf Biegeknicken und Biegedrillknicken für zweiachsige Biegung mit Normalkraft auf der Grundlage von [1].

Rückgabewert ist ein String[9] mit den ermittelten Werten für den Biegeknicknachweis und den Biegedrillknicknachweis.

Eingabeparameter:

| | | |
|---|---|---|
| Einwirkung : | - Nd in [kN]<br>- Myd in [kNm]<br>- Mzd in [kNm] | (Druck mit negativem Vorzeichen) |
| Profil: | z. B. [„I300“] | |
| Stahlguete:<br>Länge in [m] | z.B. [“St37” oder “S235”] | |
| $beta_y$: | Knicklängenbeiwert | |
| $beta_z$: | Knicklängenbeiwert | |
| $beta_y$_knicken: | Momentenbeiwert für Biegeknicken | |
| $beta_z$_knicken: | Momentenbeiwert für Biegeknicken | |
| $beta_y$_drill: | Momentenbeiwert für Biegedrillknicken | |

| | |
|---|---|
| $beta_z$_drill: | Momentenbeiwert für Biegedrillknicken |
| $z_p$ [m]: | Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt (>0 wenn Querlast am Zuggurt angreift) |
| Xi: | Momentenbeiwert für die Gabellagerung an den Stabenden |

**Bdk_Zweiachsig_MethodeII**

Die Funktion führt den Nachweis auf Biegeknicken und Biegedrillknicken für zweiachsige Biegung mit Normalkraft nach der Nachweismethode II auf der Grundlage von [1].

Rückgabewert ist ein String mit den ermittelten Werten für den Biegeknicknachweis und den Biegedrillknicknachweis in der zweiten Spalte.
Eingabe:

| | | |
|---|---|---|
| Einwirkung : | - Nd in [kN] | (Druck mit negativem Vorzeichen) |
| | - Myd in [kNm] | |
| | - Mzd in [kNm] | |
| Profil: | z. B. [„I300“] | |
| Stahlguete: | z.B. [“St37” oder “S235”] | |
| Länge in [m] | | |
| $beta_y$: | Knicklängenbeiwert | |
| $beta_z$: | Knicklängenbeiwert | |
| $beta_y$_knicken: | Momentenbeiwert für Biegeknicken | |
| $beta_z$_knicken: | Momentenbeiwert für Biegeknicken | |
| $beta_y$_drill: | Momentenbeiwert für Biegedrillknicken | |
| $beta_z$_drill: | Momentenbeiwert für Biegedrillknicken | |
| zp [m]: | Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt (>0 wenn Querlast am Zuggurt angreift) | |
| Xi: | Momentenbeiwert für die Gabellagerung an den Stabenden | |

**Bdk_Nachweis_einachsig_mitN**

Die Funktion führt den Nachweis auf Biegeknicken und Biegedrillknicken für einachsige Biegung mit Normalkraft auf der Grundlage von [1].

Rückgabewert ist ein String mit den ermittelten Werten für den Biegeknicknachweis und den Biegedrillknicknachweis.

Eingabe:

| | | |
|---|---|---|
| Einwirkung: | - $N_d$ in [kN] | (Druck mit negativem Vorzeichen) |
| | - $M_{yd}$ in [kNm] | |
| Profil: | z. B. [„I300“] | |

[9] String - Zeichenfolge

Stahlguete: z.B. [“St37” oder “S235”]

Länge in [m]

$beta_y$: Knicklängenbeiwert

$beta_z$: Knicklängenbeiwert

$beta_y$_knicken: Momentenbeiwert für Biegeknicken

$beta_y$_drill: Momentenbeiwert für Biegedrillknicken

$z_p$ [m]: Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt (>0 wenn Querlast am Zuggurt angreift)

Xi: Momentenbeiwert für die Gabellagerung an den Stabenden

**Bdk_Nachweis_einachsig_ohneN**

Die Funktion führt den Nachweis auf Biegeknicken und Biegedrillknicken für einachsige Biegung ohne Normalkraft auf der Grundlage von [1].

Rückgabewert ist ein String mit den ermittelten Werten für den Biegeknicknachweis und den Biegedrillknicknachweis.

Eingabe:

Einwirkung: - $M_{yd}$ in [kNm]

Profil: z. B. [„I300“]

Stahlguete: z.B. [“St37” oder “S235”]

Länge in [m]

$beta_y$: Knicklängenbeiwert

$beta_z$: Knicklängenbeiwert

$beta_y$_knicken: Momentenbeiwert für Biegeknicken

$beta_y$_drill: Momentenbeiwert für Biegedrillknicken

zp [m]: Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt (>0 wenn Querlast am Zuggurt angreift)

Xi: Momentenbeiwert für die Gabellagerung an den Stabenden

### 4.8.4.2 Nachweisfunktion für den Biegedrillknicknachweis in Mathcad

**biegungZweiachsig**

Die Funktion führt den Nachweis auf Biegeknicken und Biegedrillknicken für zweiachsige Biegung mit Normalkraft auf der Grundlage von [1].

Rückgabewert ist eine 1*2 Matrix mit den ermittelten Werten für den Biegeknicknachweis in der ersten Spalte und den Biegedrillknicknachweis in der zweiten Spalte.

Eingabeparameter:
Einwirkung [N|My|Mz], Profil, Stahlguete, länge [m], $beta_{Y|Z}$, $beta_{Y|Z}$_knicken, $beta_{Y|Z}$_drill, zp[m], Xi

Eingabe:

Einwirkung: $\lfloor N_d[kN] \quad M_{yd}[kNm] \quad M_{zd}[kNm] \rfloor$ 1*3 Matrix

Profil: [Profilkennung Profilhoehe] 1*2 Matrix

Profilkennung:
1 - I-Profil
2 - IPE-Profil
3 - HEA-Profil
4 - HEB-Profil
5 - HEM-Profil

Stahlguete:
1 - S235
2 - S355
3 - St37
4 - St52

Länge in [m]

$beta_{Y|Z}$: $\lfloor beta_y \quad beta_z \rfloor$ 1*2 Matrix
Knicklängenbeiwert

$beta_{Y|Z_knicken}$: $\lfloor beta_{y_knicken} \quad beta_{z_knicken} \rfloor$ 1*2 Matrix
Momentenbeiwert für Biegeknicken, nach Schneider Bautabellen 14. Version, Seite 8.48

$beta_{Y|Z_drill}$: $\lfloor beta_{y_drill} \quad beta_{z_drill} \rfloor$ 1*2 Matrix
Momentenbeiwert für Biegedrillknicken, nach Schneider Bautabellen 14. Version, Seite 8.48

zp [m]: Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt (>0 wenn Querlast am Zuggurt angreift). Auf der Biegezugseite positiv.

Xi: Momentenbeiwert für die Gabellagerung an den Stabenden, nach Schneider Bautabellen 14. Version, Seite 8.43

**biegungZweiachsigMethodeII**

Die Funktion führt den Nachweis auf Biegeknicken und Biegedrillknicken für zweiachsige Biegung mit Normalkraft nach Nachweismethode II auf der Grundlage von [1].

Rückgabewert ist eine 1*2 Matrix mit den ermittelten Werten für den Biegeknicknachweis in der ersten Spalte und den Biegedrillknicknachweis in der zweiten Spalte.

Eingabe:

Einwirkung: $\left[ N_d [kN] \quad M_{yd} [kNm] \quad M_{zd} [kNm] \right]$ 1*3 Matrix

Profil: [Profilkenn ung Profilhoeh e] 1*2 Matrix

Profilkennung:
1 - I-Profil
2 - IPE-Profil
3 - HEA-Profil
4 - HEB-Profil
5 - HEM-Profil

Stahlguete:
1 - S235
2 - S355
3 - St37
4 - St52

Länge in [m]

$beta_{Y|Z}$: $\left[ beta_y \quad beta_z \right]$ 1*2 Matrix
Knicklängenbeiwert

$beta_{Y|Z_knicken}$: $\left[ beta_{y_knicken} \quad beta_{z_knicken} \right]$ 1*2 Matrix
Momentenbeiwert für Biegeknicken, nach Schneider Bautabellen 14. Version, Seite 8.48

$beta_{Y|Z_drill}$: $\left[ beta_{y_drill} \quad beta_{z_drill} \right]$ 1*2 Matrix
Momentenbeiwert für Biegedrillknicken, nach Schneider Bautabellen 14. Version, Seite 8.48

$z_p$ [m]: Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt (>0 wenn Querlast am Zuggurt angreift). Auf der Biegezugseite positiv.

Xi: Momentenbeiwert für die Gabellagerung an den Stabenden, nach Schneider Bautabellen 14. Version, Seite 8.43

**biegungEinachsig**

Die Funktion führt den Nachweis auf Biegeknicken und Biegedrillknicken für zweiachsige Biegung mit Normalkraft nach Nachweismethode II auf der Grundlage von [1].

Rückgabewert ist eine 1*2 Matrix mit den ermittelten Werten für den Biegeknicknachweis in der ersten Spalte und den Biegedrillknicknachweis in der zweiten Spalte.

Eingabe:

Einwirkung: $\left[ N_d [kN] \quad M_{yd} [kNm] \right]$ 1*2 Matrix

Profil: [Profilkenn ung Profilhoeh e] 1*2 Matrix

Profilkennung: 1 - I-Profil
2 - IPE-Profil
3 - HEA-Profil
4 - HEB-Profil
5 - HEM-Profil

Stahlguete: 1 - S235
2 - S355
3 - St37
4 - St52

Länge in [m]

$beta_{Y|Z}$: $\left[beta_y \quad beta_z\right]$ 1*2 Matrix
Knicklängenbeiwert

$beta_{Y_knicken}$: Momentenbeiwert für Biegeknicken, nach Schneider Bautabellen 14. Version, Seite 8.48

$beta_{Y_drill}$: Momentenbeiwert für Biegedrillknicken, nach Schneider Bautabellen 14. Version, Seite 8.48

zp [m]: Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt (>0 wenn Querlast am Zuggurt angreift). Auf der Biegezugseite positiv.

Xi: Momentenbeiwert für die Gabellagerung an den Stabenden, nach Schneider Bautabellen 14. Version, Seite 8.43

**Bdk_Nachweis_einachsig_ohneN**

Die Funktion führt den Nachweis auf Biegeknicken und Biegedrillknicken für einachsige Biegung ohne Normalkraft auf der Grundlage von [1].

Rückgabewert ist eine 2*1 Matrix mit den ermittelten Werten für den Biegeknicknachweis in der ersten Spalte und den Biegedrillknicknachweis in der zweiten Spalte.

Eingabe:

Einwirkung: $M_{yd}$ in [kNm]

Profil: $\left[Profilkennung \quad Profilhoehe\right]$ 1*2 Matrix

Profilkennung: 1 - I-Profil
2 - IPE-Profil
3 - HEA-Profil
4 - HEB-Profil
5 - HEM-Profil

Stahlguete: 1 - S235
2 - S355
3 - St37
4 - St52

Länge in [m]

| | | |
|---|---|---|
| $beta_{Y\|Z}$: | $\lfloor beta_y \quad beta_z \rfloor$ Knicklängenbeiwert | 1*2 Matrix |
| $beta_{Y_knicken}$: | Momentenbeiwert für Biegeknicken, nach Schneider Bautabellen 14. Version, Seite 8.48 | |
| $beta_{Y_drill}$: | Momentenbeiwert für Biegedrillknicken, nach Schneider Bautabellen 14. Version, Seite 8.48 | |
| $z_p$ [m]: | Abstand des Angriffspunktes der Querbelastung vom Schwerpunkt (>0 wenn Querlast am Zuggurt angreift). Auf der Biegezugseite positiv. | |
| Xi: | Momentenbeiwert für die Gabellagerung an den Stabenden, nach Schneider Bautabellen 14. Version, Seite 8.43 | |

### 4.8.5 Methoden zur Ermittlung von Profilwerten

Die Bibliothek *edtra_PrfJ.dll* beinhaltet die Funktionalität zum Auslesen von Werten von normierten Walzprofilen aus dem Programm der Firma Arbed [H1]. Die Firma Arbed stellt eine Diskette zur Verfügung in denen die Werte für gebräuchliche normierte Walzprofile als ASCII-Code gespeichert sind. Die Dateien sind wie folgt definiert:

*GEOM.DAT: Enthält die geometrischen Werte der Profile und entspricht den linken Seiten des PROFILARBED Verkaufskatalogs.

*STAT.DAT: Enthält die statischen Werte der Profile und entspricht den rechten Seiten des PROFILARBED Verkaufskatalogs.

Folgende Profilserien waren auf der Diskette aufgeführt:

| | |
|---|---|
| IPE | I-Profile gemäss EU 19-57, inklusive der abgeleiteten Profile A und O sowie der Profile IPE 750 |
| HE | Breitflanschträger gemäss EU 53-62, inklusive der abgeleiteten Profile AA und der Profile HL mit extrabreiten Flanschen |
| HD | Breitflansch-Stützenprofile |
| HP | Breitflanschpfähle |
| W | Amerikanische Breitflanschträger gemäss ASTM A6/A 6M-93b |
| UB | Britische Universalträger gemäss BS4 part 1 - 1993 |
| UC | Britische Universalstützen gemäss BS4 part 1 - 1993 |
| IPN | Europäische Normalträger (Flanschneigung : 14%) |
| UAP | U-Profile mit parallelen Flanschen gemäss NF A 45-255 |
| UPN | Europäische U-Stahl Normalprofile |
| L | Gleichschenkliger Winkelstahl gemäss EU 56-77 und DIN 1028 |
| L | Ungleichschenkliger Winkelstahl gemäss EU 57-78 und DIN 1029 |

Ausgegeben werden folgende Werte:

| | |
|---|---|
| Bezeichnung | Bezeichnung des Profils |
| G | Metergewicht in kg/m |
| h | Höhe in mm |
| b | Breite in mm |
| tw | Stegdicke in mm |
| tf | Flanschdicke in mm |
| r | Ausrundungsradius in mm |
| A | Querschnittsfläche in cm2 |
| hi | Kammerhöhe in mm |
| d | Höhe zwischen Ausrundungen in mm |
| | maximaler Schraubendurchmesser |
| pmin | minimaler Schraubenabstand senkrecht zur Kraftrichtung in mm |
| pmax | maximaler Schraubenabstand senkrecht zur Kraftrichtung in mm |
| AL | Anstrichfläche pro Meter in m2 |
| AG | Anstrichfläche pro Tonne in m2 |

Darüberhinaus sind eventuell für Winkel und U-Profile angegeben:

| | |
|---|---|
| r2 | Abrundungsradius in mm |
| v1 u. v2 | Distanz der äusseren Faser zur u-Achse in mm |
| u1, u2 u. u3 | Distanz der äusseren Fasern zur v-Achse in mm |
| zS | Schwerpunktsabstand in Richtung z-Achse in cm |
| yS | Schwerpunktsabstand in Richtung y-Achse in cm |

Zudem wird ausgegeben:

| | |
|---|---|
| G | Metergewicht in kg/m |
| Iy | Flächenmoment 2. Grades um die y-Achse in cm4 |
| Wy | elastisches Widerstandsmoment um die y-Achse in cm3 |
| Wpl,y | plastisches Widerstandsmoment um die y-Achse in cm3 |
| iy | Trägheitshalbmesser bezogen auf die y-Achse in cm |
| Avz | Querkraftfl„che des Stegs in cm2 |
| Iz | Flächenmoment 2. Grades um die z-Achse in cm4 |
| Wz | elastisches Widerstandsmoment um die z- Achse in cm3 |
| Wpl,z | plastisches Widerstandsmoment um die z-Achse in cm3 |
| iz | Trägheitshalbmesser bezogen auf die z-Achse in cm |
| Ss | steife Auflagerlänge in mm |
| IT | Torsionsflächenmoment 2. Grades in cm4 |
| Iw x10E-3 | Wölbflächenmoment 2. Grades in cm6 |

Einstufung in Querschnittsklassen nach ENV 1993-1-1:

reine Biegung für S235
reine Biegung für S355
reine Biegung für S460
reine Druckbeanspruchung für S235
reine Druckbeanspruchung für S355
reine Druckbeanspruchung für S460

Sowie eine Angabe ob das Profil in HISTAR-Güte geliefert werden kann (1) oder nicht (0).

Darüber hinaus sind für U-Profile angegeben

yS Schwerpunktsabstand in Richtung y-Achse in cm
yM Abstand des Schubmittelpunkts in Richtung y-Achse in cm

Für Winkelstähle
Iu u. Iv Flächenmomente 2. Grades der Hauptträgheitsachsen in cm4
Iyz Flächenzentrifugalmoment 2. Grades in cm4
à Neigung der Hauptträgheitsachsen in Grad

Hierfür wurden 17 Funktionen für Mathcad und eine Funktionen für MS-Excel entwickelt. Diese werden nachfolgend erläutert.

Für Excel wurde hier die Möglichkeit des Aufrufs über die Menüleiste geschaffen.
Extras->ED_TRA Funktionen->Profiltafeln.

Für Mathcad ergibt sich wieder die in Kapitel 4.5.3 bereits erläuterte Vorgehensweise bei der Eingabe. Über die Menüleiste wird das Dialogfenster *Funktion einfügen* aufgerufen und die gewünschte Funktion ausgewählt. Die Funktionen sind in der Kategorie *Profiltabellen* zu finden Hierbei ergibt sich wieder der Umstand, dass in Mathcad keine Zeichenfolgen eingegeben werden können, was die große Anzahl der erstellten Funktionen erklärt. Es wurde für jede Profilreihe eine eigene Funktion erstellt.

| **Funktionen in MS-Excel** | **Funktionen in Mathcad** |
|---|---|
| Profiltafeln | I_Profil |
| | IPE_Profil |
| | Sonderprofil_IPE750 |
| | IPN_Profil |
| | L_Profil |
| | W_Profil |
| | UB_Profil |
| | UC_Profil |
| | UPN_Profil |
| | U_Profil |
| | UAP_Profil |
| | HP_Profil |
| | HL_Profil |
| | Sonderprofil_HL |
| | HE_Profil |
| | Sonderprofil_HE |
| | HD_Profil |

Tabelle 21. Funktionen für die Ermittlung von Profilwerten

#### 4.8.5.1 Funktion Profiltafeln in Excel

Der Aufruf der Funktion erfolgt über die Menüleiste
(Extras->ED_TRA Funktionen->Profiltafeln)

Die Eingabesyntax für die Profile ist die Normbezeichnung mit anschließender Profilhöhe ohne Leerzeichen dazwischen.
Bsp:
IPE300 oder HEA200
Die Rückgabewerte entsprechen den oben angegebenen Werten für die jeweiligen Profile und können explizit aus folgendem Kapitel entnommen werden.

#### 4.8.5.2 Funktionen in Mathcad

**I_Profil**

Die Fuktion liefert die geometrischen und statischen Werte für I-Profile in einer 1*n Matrix. Die Werte in ihrer Reihenfolge sind unten aufgeführt.
Eingabewerte:
Profilhoehe in [mm]

| | | | |
|---|---|---|---|
| 1. | G | Gewicht pro Meter | [kg/m] |
| 2. | h | Hoehe=Breite des Profils | [mm] |
| 3. | b | Breite des Profils | [mm] |
| 4. | tw | Stegdicke | [mm] |
| 5. | tf | Flanschdicke | [mm] |
| 6. | r1 | Abrundungsradius | [mm ] |
| 7. | r2 | Abrundungsradius | [mm ] |
| 8. | A | Querschnittsflaeche | [cm^2] |
| 9. | d | Hoehe zwischen Ausrundungen | [cm] |
| 10. | i | maximaler Schraubendurchmesser | [mm] |
| 11. | pmin | minimaler Schraubenabstand senkrecht zur Kraftrichtung | [mm] |
| 12. | pmax | maximaler Schraubenabstand senkrecht zur Kraftrichtung | [mm] |
| 13. | AL | Anstrichflaeche pro Meter | [m2/m] |
| 14. | G | Anstrichflaeche pro Tonne | [m2/t] |
| 15. | G | Gewicht pro Meter | [kg/m] |
| 16. | Iy | Flaechenmoment 2. Grades | [cm^4] |
| 17. | Wy | elast. Widerstandsmoment | [cm^3] |
| 18. | Wpl,y | plast. Widerstandsmoment | [cm^3] |
| 19. | iy | Traegheitshalbmesser | [cm] |
| 20. | Avz | Querkraftflaeche des Stegs | [cm^2] |
| 21. | Iz | Flaechenmoment 2. Grades | [cm^4] |
| 22. | Wz | elast. Widerstandsmoment | [cm^3] |
| 23. | Wplz | plast. Widerstandsmoment | [cm^3] |
| 24. | iz | Traegheitshalbmesser | [cm] |
| 25. | Ss | steife Auflagerlaenge | [mm] |
| 26. | It | Torsionsflaechenmoment 2. Grades | [cm^4] |
| 27. | Iw | Woelbflaechenmoment 2. Grades | [cm^6] |

Einstufung in Querschnittsklassen nach ENV 1993-1-1:
28. reine Biegung fuer S235
29. reine Biegung fuer S355
30. reine Druckbeanspruchung fuer S235
31. reine Druckbeanspruchung fuer S355
32. HISTAR Guete lieferbar 0.00 = Nein ; 1.00= Ja

**IPE_Profil**

Die Fuktion liefert die geometrischen und statischen Werte für IPE-Profile in einer 1*n Matrix.
Die Werte in ihrer Reihenfolge sind unten aufgeführt.
Eingabewerte:
Profilhoehe in [mm]

| | | | |
|---|---|---|---|
| 1. | G | Gewicht pro Meter | [kg/m] |
| 2. | h | Hoehe=Breite des Profils | [mm] |
| 3. | b | Breite des Profils | [mm] |
| 4. | tw | Stegdicke | [mm] |
| 5. | tf | Flanschdicke | [mm] |
| 6. | r1 | Abrundungsradius | [mm ] |
| 7. | r2 | Abrundungsradius | [mm ] |
| 8. | A | Querschnittsflaeche | [cm^2] |
| 9. | d | Hoehe zwischen Ausrundungen | [cm] |
| 10. | i | maximaler Schraubendurchmesser | [mm] |
| 11. | pmin | minimaler Schraubenabstand senkrecht zur Kraftrichtung | [mm] |
| 12. | pmax | maximaler Schraubenabstand senkrecht zur Kraftrichtung | [mm] |
| 13. | AL | Anstrichflaeche pro Meter | [m2/m] |
| 14. | G | Anstrichflaeche pro Tonne | [m2/t] |
| 15. | G | Gewicht pro Meter | [kg/m] |
| 16. | Iy | Flaechenmoment 2. Grades | [cm^4] |
| 17. | Wy | elast. Widerstandsmoment | [cm^3] |
| 18. | Wpl,y | plast. Widerstandsmoment | [cm^3] |
| 19. | iy | Traegheitshalbmesser | [cm] |
| 20. | Avz | Querkraftflaeche des Stegs | [cm^2] |
| 21. | Iz | Flaechenmoment 2. Grades | [cm^4] |
| 22. | Wz | elast. Widerstandsmoment | [cm^3] |
| 23. | Wplz | plast. Widerstandsmoment | [cm^3] |
| 24. | iz | Traegheitshalbmesser | [cm] |
| 25. | Ss | steife Auflagerlaenge | [mm] |
| 26. | It | Torsionsflaechenmoment 2. Grades | [cm^4] |
| 27. | Iw | Woelbflaechenmoment 2. Grades | [cm^6] |

Einstufung in Querschnittsklassen nach ENV 1993-1-1:
28. reine Biegung fuer S235
29. reine Biegung fuer S355
30. reine Biegung fuer S460
31. reine Druckbeanspruchung fuer S235
32. reine Druckbeanspruchung fuer S355
33. reine Druckbeanspruchung fuer S460
34. HISTAR Guete lieferbar 0.00 = Nein ; 1.00= Ja

**Sonderprofil_IPE750**
Eingabewerte:
Profilhoehe in [mm]

Die Fuktion liefert die geometrischen und statischen Werte für IPE-Profile (Sonderprofil) in einer 1*n Matrix. Die Werte in ihrer Reihenfolge sind unten aufgeführt.

Die Ausgabewerte entsprechen den Werten der Funktion IPE_Profil

**IPN_Profil**

Eingabewerte:
Profilhoehe in [mm]

Die Fuktion liefert die geometrischen und statischen Werte für I-Profile in einer 1*n Matrix. Die Werte in ihrer Reihenfolge sind unten aufgeführt.

Die Ausgabewerte entsprechen den Werten der Funktion I_Profil

**L_Profil**

L_Profil(Schenkel_1 [mm],Schenkel_2 [mm],Blechstaerke [mm])

Die Fuktion liefert die geometrischen und statischen Werte für gleichschenklige und ungleichsxchenklige L-Profile in einer 1*n Matrix. Die Werte in ihrer Reihenfolge sind unten aufgeführt.

Gleichschenkliger Winkelstahl gemaess EN 56-77 und DIN 1028

| | | | |
|---|---|---|---|
| 1. | G | Gewicht pro Meter | [kg/m] |
| 2 | h=b | Hoehe=Breite des Profils | [mm] |
| 3. | t | Materialstaerke | [mm] |
| 4. | r1 | Flanschdicke | [mm] |
| 5. | r2 | Abrundungsradius | [mm] |
| 6. | A | Querschnittsflaeche | [cm^2] |
| 7 | zs=ys | Schwerpunktsabstand z\|y | [cm] |
| 8. | v | Distanz der aeusseren Faser zur v-Achse | [cm] |
| 9. | u1 | Distanz der aeusseren Fasern | |
| 10. | u2 | zur v-Achse | [cm] |
| 11. | AL | Anstrichflaeche pro Meter | [m2/m] |
| 12. | AG | Anstrichflaeche pro Tonne | [m2/t] |
| 13. | G | Gewicht pro Meter | [kg/m] |
| 14. | Iy=Iz | Flaechenmoment 2. Grades | [cm^4] |
| 15. | Wy=Wz | elast. Widerstandsmoment | [cm^3] |
| 16. | iy=iz | Traegheitshalbmesser | [cm] |
| 17. | Iu | Hauptflaechenmoment 2. Gr | [cm^4] |
| 18. | iu | | |
| 19. | Iv | Hauptflaechenmoment 2. Gr | [cm^4] |
| 20. | iv | | |
| 21. | Iyz | Flaechenzentrifugalmoment 2. Grades | [cm^4] |

Einstufung in Querschnittsklassen nach ENV 1993-1-1:
22. reine Druckbeanspruchung fuer S235
23. reine Druckbeanspruchung fuer S355
24. HISTAR Guete lieferbar 0.00 = Nein ; 1.00= Ja

Ungleichschenkliger Winkelstahl gemaess EN 57-78 und DIN 1029

| | | | |
|---|---|---|---|
| 1. | G | Gewicht pro Meter | [kg/m] |
| 2 | h | Hoehe des Profils | [mm] |
| 3 | b | Breite des Profils | [mm] |
| 4. | t | Materialstaerke | [mm] |
| 5. | r1 | Flanschdicke | [mm] |
| 6. | r2 | Abrundungsradius | [mm ] |
| 7. | A | Querschnittsflaeche | [cm^2] |
| 8. | zs | Schwerpunktsabstand z | [cm] |
| 9. | ys | Schwerpunktsabstand y | [cm] |
| 10. | v | Distanz der aeusseren Faser zur v-Achse | [cm] |
| 11. | u1 | Distanz der aeusseren Fasern | |
| 12. | u2 | zur v-Achse | [cm] |
| 13. | u3 | | [cm] |
| 14. | AL | Anstrichflaeche pro Meter | [m2/m] |
| 15. | AG | Anstrichflaeche pro Tonne | [m2/t] |
| 16. | G | Gewicht pro Meter | [kg/m] |
| 17 | Iy | Flaechenmoment 2. Grades | [cm^4] |
| 18. | Wy | elast. Widerstandsmoment | [cm^3] |
| 19. | iy | Traegheitshalbmesser | [cm] |
| 20. | Iz | Flaechenmoment 2. Grades | [cm^4] |
| 21. | Wz | elast. Widerstandsmoment | [cm^3] |
| 22. | iz | Traegheitshalbmesser | [cm] |
| 23. | Iu | Hauptflaechenmoment 2. Gr. | [cm^4] |
| 24. | iu | | |
| 25. | Iv | Hauptflaechenmoment 2. Gr. [cm^4] | |
| 26. | iv | | |
| 27. | Iyz | Flaechenzentrifugalmoment 2. Grades | [cm^4] |
| 28. | à | - Neigung der Haupttraegheitsachsen | [Grad] |

Einstufung in Querschnittsklassen nach ENV 1993-1-1:
29. reine Druckbeanspruchung fuer S235
30. reine Druckbeanspruchung fuer S355
31. HISTAR Guete lieferbar 0.00 = Nein ; 1.00= Ja

**W_Profil**

W_Profil(Hoehe[mm],Breite[mm],Gewicht[kg/m])

Die Fuktion liefert die geometrischen und statischen Werte für W-Profile (Amerikanische Breitflanschprofile)in einer 1*n Matrix.
Die Werte in ihrer Reihenfolge sind unten aufgeführt.

Die Ausgabewerte entsprechen den Werten der Funktion I_Profil

**UB_Profil**

UB_Profil(Hoehe[mm],Breite[mm],Gewicht[kg/m])

Die Fuktion liefert die geometrischen und statischen Werte für UB-Profile (Britische Universaltraeger gemaess BS4 part 1 - 1993) in einer 1*n Matrix.

Die Ausgabewerte entsprechen den Werten der Funktion IPE_Profil

**UC_Profil**

UC_Profil(Hoehe[mm],Breite[mm],Gewicht[kg/m])

Die Fuktion liefert die geometrischen und statischen Werte für UC-Profile (Britische Universalstuetzen gemaess BS4 part 1 - 1993) in einer 1*n Matrix.

**UPN_Profil**

UPN_Profil(Hoehe[mm])

Die Fuktion liefert die geometrischen und statischen Werte für UC-Profile (Britische Universalstuetzen gem„ss BS4 part 1 - 1993) in einer 1*n Matrix. Die Werte in ihrer Reihenfolge sind unten aufgeführt.

| | | | |
|---|---|---|---|
| 1. | G | Gewicht pro Meter | [kg/m] |
| 2 | h | Hoehe=Breite des Profils | [mm] |
| 3 | b | Breite des Profils | [mm] |
| 4. | s | Materialstaerke | [mm] |
| 5. | t=r1 | Flanschdicke | [mm] |
| 6. | r2 | Abrundungsradius | [mm ] |
| 7. | A | Querschnittsflaeche | [cm^2] |
| 8. | h1 | Kammerhoehe | [mm] |
| 9. | G | Gewicht pro Meter | [kg/m] |
| 10. | Iy | Flaechenmoment 2. Grades | [cm^4] |
| 11. | Wy | elast. Widerstandsmoment | [cm^3] |
| 12. | Wpl,y | plast. Widerstandsmoment | [cm^3] |
| 13. | iy | Traegheitshalbmesser | [cm] |
| 14. | Avz | Querkraftflaeche des Stegs | [cm^2] |
| 15. | Iz | Flaechenmoment 2. Grades | [cm^4] |
| 16. | Wz | elast. Widerstandsmoment | [cm^3] |
| 17. | Wplz | plast. Widerstandsmoment | [cm^3] |
| 18. | iz | Traegheitshalbmesser | [cm] |
| 19. | Ss | steife Auflagerlaenge | [mm] |
| 20. | It | Torsionsflaechenmoment 2. Grades | [cm^4] |
| 21. | Iw | Woelbflaechenmoment 2. Grades | [cm^6] |

Einstufung in Querschnittsklassen nach ENV 1993-1-1:
22. reine Biegung fuer S235
23. reine Biegung fuer S355
24. reine Druckbeanspruchung fuer S235
25. reine Druckbeanspruchung fuer S355
26. HISTAR Guete lieferbar 0.00 = Nein ; 1.00= Ja

### U_Profil

U_Profil(Hoehe[mm])

Die Fuktion liefert die geometrischen und statischen Werte für U-Profile nach DIN 1026 (10.63) in einer 1*n Matrix.

Die Ausgabewerte entsprechen den Werten der Funktion UPN_Profil

### UAP_Profil

UAP_Profil(Hoehe[mm])

Die Fuktion liefert die geometrischen und statischen Werte für UAP-Profile (U-Profile mit parallelen Flanschen gemaess NF A 45-255) in einer 1*n Matrix.

Die Ausgabewerte entsprechen den Werten der Funktion UPN_Profil

### HP_Profil

HP_Profil(Profilhoehe, Gewicht)

Die Fuktion liefert die geometrischen und statischen Werte für HP-Profile (Breitflanschpfaehle) in einer 1*n Matrix.

Eingabewerte:
Profilhoehe in [mm]
Profilgewicht [kg/m]

Die Ausgabewerte entsprechen den Werten der Funktion IPE_Profil

### HE_Profil

HE_Profil(Profilhoehe, Profilart)

Eingabewerte:
Profilhoehe in [mm]
Profilart:
1 - HEA-Profil
2 - HEAA-Profil
3 - HEB-Profil
4 - HEM-Profil

Die Funktion liefert die geometrischen und statischen Werte für HE-Profile (Breitflanschtraeger gemaess EN 53-62) in einer 1*n Matrix.

Die Ausgabewerte entsprechen den Werten der Funktion IPE_Profil

**Sonderprofil_HE**

Sonderprofil_HE(Profilhoehe, Gewicht)

Eingabewerte:
Profilhoehe in [mm]
Profilgewicht [kg/m]

Die Profilreihe HE Breitflanschtraeger gemaess EN 53-62) Profile HE AA bietet manche Profile in mehreren Steg und Flanschstärken an. Diese unterscheiden sich hauptsächlich in den Flansch und Stegstärken des Blechs. Die verschiedenen Profile:
HE 600 x 340
HE 600 x 402
HE 650 x 347
HE 650 x 410
HE 700 x 356
HE 700 x 421
HE 800 x 377
HE 800 x 448
E 900 x 396
HE 900 x 471
HE 1000 x 415
HE 1000 x 494
Für den zweiten Wert wird lediglich die zweite Zahl hinter dem x eingegeben

Die Ausgabewerte entsprechen den Werten der Funktion IPE_Profil

**HL_Profil**

HL_Profil(Profilhoehe, Profilart)

Die Fuktion liefert die geometrischen und statischen Werte für HL-Profile in einer 1*n Matrix. Breitflanschtraeger gemaess EN 53-62, von HE-Profilen abgeleitetes Profil der Reihe HL 1000 und HL 1100 mit extrabreiten Flanschen
Eingabewerte:
Profilhoehe in [mm]
Profilart:
1 - HLA-Profil
2 - HLB-Profil
3 - HLM-Profil
4 - HLR-Profil

Es existieren lediglich die unten stehenden Profile in den Tabellen
HL 1000 A
HL 1000 B
HL 1000 M
HL 1100 A
HL 1100 B
HL 1100 M
HL 1100 R

Die Ausgabewerte entsprechen den Werten der Funktion IPE_Profil

**Sonderprofil_HL**

Sonderprofil_HL(Profilhoehe, Gewicht)

Die Fuktion liefert die geometrischen und statischen Werte für HL-Profile (Sonderprofil) in einer 1*n Matrix.

Eingabewerte:
Profilhoehe in [mm]

Profilgewicht [kg/m]
Die Profilreihe HL (Breitflanschtraeger gemaess EN 53-62) bietet manche Profile in mehreren Steg und Flanschstärken an. Diese unterscheiden sich hauptsächlich in den Flansch und Stegstärken des Blechs. Die vier verschiedenen Profile:

HL 1000 x 296
HL 1000 x 477
HL 1000 x 554
HL 1000 x 642
Für den zweiten Wert wird lediglich die zweite Zahl hinter dem x eingegeben

Die Ausgabewerte entsprechen den Werten der Funktion IPE_Profil

**HD_Profil**

HD_Profil(Schenkel_1[mm], Schenkel_2[mm])

Die Fuktion liefert die geometrischen und statischen Werte für HD-Profile (Breitflansch Stützenprofile) in einer 1*n Matrix.

Die Ausgabewerte entsprechen den Werten der Funktion IPE_Profil

## 4.9 Holzbau

### 4.9.1 Allgemeines

Im Rahmen des Holzbaus wurden 17 Funktionen entwickelt, die nachfolgend erläutert werden.

Da diese Berechnungen keine umfangreicheren Eingaben benötigen, war für Excel keine eigene Oberfläche erforderlich und es konnte auf die Standardmöglichkeit der Eingabe zurückgegriffen werden (siehe hierzu auch Kapitel II-1.4.1.1).

Für Mathcad ergibt sich wieder die in Kapitel 4.5.3 bereits erläuterte Vorgehensweise bei der Eingabe. Über die Menüleiste wird das Dialogfenster *Funktion einfügen* aufgerufen und die gewünschte Funktion ausgewählt. Die Funktionen sind in der Kategorie *Holzbau* zu finden Hierbei ergibt sich wieder der Umstand, dass in Mathcad keine Zeichenfolgen eingegeben werden können. So muss hier für die verschiedenen Holzsorten eine Nummer eingegeben

werden, die jedoch in folgender Tabelle angegeben werden und auf die bei den Funktionen verwiesen wird.
Zum Beispiel: 1 - "NH_S7” oder 10 - "BS14_S13"

Die Ziffer vor der Bezeichnung in Anführungsstrichen entspricht der Nummer, die in Mathcad für die Holzgüte zu verwenden ist. Siehe auch folgende Tabelle 22.

| | |
|---|---|
| 1 | "NH_S7” |
| 2 | "NH_S10” |
| 3 | "NH_S13" |
| 4 | "NH_MS13" |
| 5 | "NH_MS17" |
| 6 | "LH_A" |
| 7 | "LH_B" |
| 8 | "LH_C" |
| 9 | “BS11_S10" |
| 10 | "BS14_S13" |
| 11 | "BS16_MS13" |
| 12 | "BS18_MS17" |

Tabelle 22. Bezeichnungen für die Holzgüte in Mathcad und MS-Excel

### 4.9.2 Normalkraftbeanspruchung

Für die Normalkraftbeanspruchung im Holzbau wurden folgende Funktionen umgesetzt:

HolzZugNachweisV1

HolzZugNachweisV2

HolzZug_zulN_V1

HolzZug_zulN_V2

HolzZug_erfBreite

HolzZug_erfQuerschnitt

HolzDruck_Nachweis

HolzDruck_zul_N

HolzDruck_Bem

Diese Funktionen werden im Folgenden weiter erläutert.

**HolzZugNachweisV1**

Die Funktion liefert den Nachweis eines Zugstabes.
Eingabeparameter: Normalkraft, Breite, Höhe, Holzart [1-12]
Eingabe:
N - Normalkraft [kN]
b - Querschnittsbreite [cm]
h - Querschnittshöhe [cm]
Holzart - Holzartsortierklasse

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzZugNachweisV2**

Die Funktion liefert den Nachweis eines Zugstabes. Der Unterschied zur Funktion *HolzZugNachweisV1* ist die Eingabe der Querschnittsfläche anstatt der Breite und Höhe des Querschnitts.

Eingabeparameter: N[kN], A[cm²],, Holzart[1-12]

Eingabe:
N - Normalkraft [kN]
A - Querschnitt [cm²]
Holzart – Holzartsortierklasse

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzZug_zulN_V1**

Die Funktion liefert die zul. Normalkraft [kN] eines Zugstabes.

Eingabeparameter: b[cm],h[cm], Holzart[1-12]

Eingabe:
b - Querschnittsbreite [cm]
h - Querschnittshöhe [cm]
Holzart – Holzartsortierklasse

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzZug_zulN_V2**

Die Funktion liefert die zul. Normalkraft [kN] eines Zugstabes. Der Unterschied zur Funktion *HolzZug_zulN_V1* ist die Eingabe der Querschnittsfläche anstatt der Breite und Höhe des Querschnitts.

Eingabeparameter: A[$cm^2$], Holzart[1-12]

Eingabe:
A - Querschnitt [cm²]
Holzart - Holzartsortierklasse

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

### HolzZug_erfBreite

Die Funktion liefert die erforderliche Breite [cm] eines Zugstabes.

Eingabeparameter: N[kN],h[cm],Holzart[1-12]

Eingabe:

| | |
|---|---|
| N | - Normalkraft [kN] |
| h | - Querschnittshöhe [cm] |
| Holzart | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

### HolzZug_erfQuerschnitt

Die Funktion liefert den erforderlichen Querschnitt [cm²] eines Zugstabes.

Eingabeparameter: N[kN], Holzart[1-12]

Eingabe:

| | |
|---|---|
| N | - Normalkraft [kN] |
| Holzart | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

### HolzDruck_Nachweis

Die Funktion liefert den Nachweis eines Druckstabes.

Eingabeparameter: N[kN],sky[cm],skz[cm],Holzart[1-12],b[cm],h[cm]

Eingabe:

| | |
|---|---|
| N | - Normalkraft [kN] |
| sky | - Knicklänge y |
| skz | - Knicklänge z |
| b | - Querschnittsbreite [cm] |
| h | - Querschnittshöhe [cm] |
| Holzart | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzDruck_zul_N**

Die Funktion liefert die zulässige Normalkraft eines Druckstabes.

Eingabeparameter: sky[cm],skz[cm],b[cm],h[cm] ,Holzart[1-12]

Eingabe:

| | |
|---|---|
| sky | - Knicklänge y |
| skz | - Knicklänge z |
| b | - Querschnittsbreite [cm] |
| h | - Querschnittshöhe [cm] |
| Holzart | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzDruck_Bem**

Die Funktion liefert die erforderliche Breite eines Druckstabes

Eingabeparameter: N[kN], sky[cm],skz[cm],h[cm] ,Holzart[1-12]

Eingabe:

| | |
|---|---|
| N | - Normalkraft [kN] |
| sky | - Knicklänge y |
| skz | - Knicklänge z |
| h | - Querschnittshöhe [cm] |
| Holzart | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

### 4.9.3 Biegung

Für die Biegebeanspruchung im Holzbau wurden folgende Funktionen umgesetzt:

HolzDruckMyMzNachweis

HolzDruckMNachweis

HolzZugMyMzNachweis

HolzZugMNachweis

HolzDruckMyMzErfBreite

HolzDruckMErfBreite

HolzZugMyMzErfBreite

HolzZugMyErfBreite

Die Funktionen werden nachfolgend kurz erläutert

**HolzDruckMyMzNachweis**

Die Funktion liefert den Nachweis für zweiachsige Biegung und Druck.

Eingabeparameter: My[kNm],Mz[kNm],N[KN],s[m],sKy[m],sKz[m],h[m],b[m],HolzKl

Eingabe:

| | |
|---|---|
| My | - Moment [kNm] |
| Mz | - Moment [kNm] |
| N | - Normalkraft [kN] |
| s | - seitliche Abstützung[m] |
| sky | - Knicklänge[m] |
| skz | - Knicklänge[m] |
| h | - Querschnittshöhe[m] |
| b | - Querschnittsbreite[m] |
| HolzKl | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzDruckMNachweis**

Die Funktion liefert den Nachweis Biegung und Druck.

Eingabeparameter: My[kNm],N[KN],s[m],sKy[m],sKz[m],h[m],b[m],HolzKl

Eingabe:

| | |
|---|---|
| My | - Moment [kNm] |
| N | - Normalkraft [kN] |
| s | - seitliche Abstützung[m] |
| sky | - Knicklänge[m] |
| skz | - Knicklänge[m] |
| h | - Querschnittshöhe[m] |
| b | - Querschnittsbreite[m] |
| HolzKl | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzZugMyMzNachweis**

Die Funktion liefert den Nachweis für zweiachsige Biegung und Zug.
Eingabeparameter: My[kNm],Mz[kNm],N[KN],h[m],b[m],HolzKl

Eingabe:

| | |
|---|---|
| My | - Moment [kNm] |
| Mz | - Moment [kNm] |

N - Normalkraft [kN]
h - Querschnittshöhe[m]
b - Querschnittsbreite[m]
HolzKl - Holzartsortierklasse

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzZugMNachweis**

Die Funktion liefert den Nachweis für Biegung und Zug.
Eingabeparameter: My[kNm],N[KN],h[m],b[m],HolzKl

Eingabe:
My - Moment [kNm]
N - Normalkraft [kN]
h - Querschnittshöhe[m]
b - Querschnittsbreite[m]
HolzKl - Holzartsortierklasse

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzDruckMyMzErfBreite**

Die Funktion liefert die erforderliche Breite[cm] eines Biegedruckstabes mit zweiachsiger Biegung.

Eingabeparameter: My[kNm],Mz[kNm],N[Kn],s[m],sKy[m],sKz[m],h[m],HolzKl

Eingabe:
My - Moment [kNm]
Mz - Moment [kNm]
N - Normalkraft [kN]
s - seitliche Abstützung[m]
sky - Knicklänge[m]
skz - Knicklänge[m]
h - Querschnittshöhe[m]
HolzKl - Holzartsortierklasse

**HolzDruckMErfBreite**

Die Funktion liefert die erforderliche Breite[cm] eines Biegedruckstabes mit einachsiger Biegung.

Eingabeparameter: My[kNm],N[Kn],s[m],sKy[m],sKz[m],h[m],HolzKl

Eingabe:

| | |
|---|---|
| My | - Moment [kNm] |
| N | - Normalkraft [kN] |
| s | - seitliche Abstützung[m] |
| sky | - Knicklänge[m] |
| skz | - Knicklänge[m] |
| h | - Querschnittshöhe[m] |
| HolzKl | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzZugMyMzErfBreite**

Die Funktion liefert die erforderliche Breite[cm] eines Zugstabes mit zweiachsiger Biegung.

Eingabeparameter: My[kNm],Mz[kNm],N[KN],h[m],HolzKl
Eingabe:

| | |
|---|---|
| My | - Moment [kNm] |
| Mz | - Moment [kNm] |
| N | - Normalkraft [kN] |
| h | - Querschnittshöhe[m] |
| HolzKl | - Holzartsortierklasse |

Bemerkung:
Es stehen die in Tabelle 22 angegebenen Holzartsortierklassen zur Verfügung. Bitte die jeweilige Bezeichnung eingeben.

**HolzZugMyMzErfBreite**

Die Funktion liefert die erforderliche Breite[cm] eines Zugstabes mit einachsiger Biegung.
Eingabeparameter: My[kNm],N[KN],h[m],HolzKl

Eingabe:

| | |
|---|---|
| My | - Moment [kNm] |
| N | - Normalkraft [kN] |
| h | - Querschnittshöhe[m] |
| HolzKl | - Holzartsortierklasse |

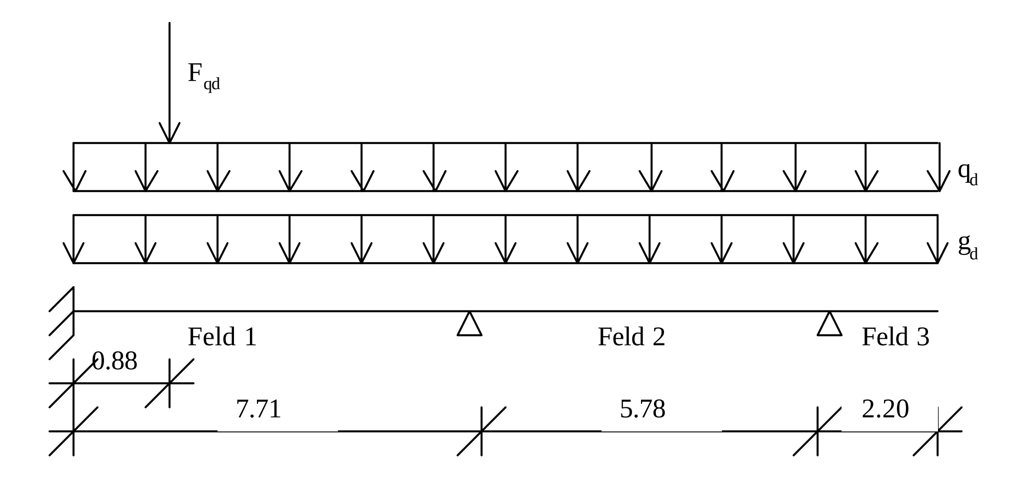
**Literatur zu Kapitel II-4**

[1] Schneider
„Bautabellen für Ingenieure“
13. Auflage, Werner Verlag, 1998

[H1] EUROPROFIL S.A.
Societe, commerciale de
PROFILARBED
66, rue de Luxembourg
4009 Esch-sur-/Alzette
Luxembourg

# II-5 Beispiele

## 5.1 Durchlaufträger in Stahlbeton nach DIN 1045-1




**Systemwerte:**

Feldlängen [m]

$L_1 := 7.71$

$L_2 := 5.78$

$L_3 := 2.20$

Abschnittslängen [m]

$l_{a1} := 0.88$

$l_{a2} := 6.83$

$l_{a3} := 5.78$

$l_{a4} := 2.20$

Querschnittswerte:

Breite in m $b := 0.3$

Rohdichte $:= 25$ kN/m³

Höhe in m $h := 0.5$

Trägheitsmodul $I := b \cdot \frac{h^3}{12}$ $I = 0.003\ m^4$

Material: - Beton C35/45

Druckfestigeit in N/mm² $f_{ck} := 35$

- Betonstahl BSt 500 S

Nennstreckgrenze in N/mm² $f_{yk} := 500$

Elastizitätsmodul in kN/m² $E := 33300000$

$E \cdot I = 1.041 \times 10^5$ kN/m²

**Belastungszusammenstellung**

Teilsicherheitsbeiwerte $\gamma_G := 1.35$ $\gamma_Q := 1.50$

Ständige Lasten: -Eigengewicht in kN/m $g_E := b \cdot h \cdot \text{Rohdichte}$ $g_E = 3.75$

-sonstige ständigeLasten in kN/m $g_s := 10$

Summe der ständigen Lasten in kN/m $g_k := g_E + g_s$ $g_k = 13.75$

Verkehrslast: Streckenlast in kN/m $q_k := 20$ Einzellast in kN $F_{qk} := 100$

Zur Ermittlung der Schnittgrößen im Grenzzustand der Tragfähigkeit werden die charakteristischen Werte der Einwirkung mit den Teilsicherheitsbeiwerten, die von der Lastart und dem betrachteten Grenzzustand abhängen, multipliziert:

$g_d := \gamma_G \cdot g_k$ $q_d := \gamma_Q \cdot q_k$ $F_{qd} := \gamma_Q \cdot F_{qk}$

$g_d = 18.563$ $q_d = 30$ $F_{qd} = 150$

**Eingabewerte für die DLT Funktionen**

Zeilenweise Eingabe der Abschnitte
(Feld_Nummer, Abschnitts_Nummer, Abschnitts_Länge[m])

$$Ab := \begin{pmatrix} 1 & 1 & l_{a1} \\ 1 & 2 & l_{a2} \\ 2 & 1 & l_{a3} \\ 3 & 1 & l_{a4} \end{pmatrix}$$

Feldweise Eingabe der Querschnittsdefinitionen [kN*m^2]

$EI := (E \cdot I \quad E \cdot I \quad E \cdot I)$

Feldweise Eingabe des Bettungsmodul [kN/m^3]

$Cs := (0 \quad 0 \quad 0)$

Feldweise Eingabe der Normalkraft Th.II Ordnung [kN]
N = 0 ( Th. I Ordnung )

$N := \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$

Lagerdefinition am linken Trägerrand

$Lli := 2$

Lagerdefinition am rechten Trägerrand

$Lre := 3$

Zeilenweise Eingabe der Abschnittslasten
Feld_Nummer, Abschnitts_Nummer, Lastfall_Nummer, q_li [kN/m], q_re [kN/m]

$$Ab_LastTragf := \begin{pmatrix} 1 & 1 & 1 & g_d & g_d \\ 1 & 2 & 1 & g_d & g_d \\ 2 & 1 & 1 & g_d & g_d \\ 3 & 1 & 1 & g_d & g_d \\ 1 & 1 & 2 & q_d & q_d \\ 1 & 2 & 2 & q_d & q_d \\ 2 & 1 & 3 & q_d & q_d \\ 3 & 1 & 4 & q_d & q_d \end{pmatrix} \qquad Ab_LastGebr := \begin{pmatrix} 1 & 1 & 1 & g_k & g_k \\ 1 & 2 & 1 & g_k & g_k \\ 2 & 1 & 1 & g_k & g_k \\ 3 & 1 & 1 & g_k & g_k \\ 1 & 1 & 2 & q_k & q_k \\ 1 & 2 & 2 & q_k & q_k \\ 2 & 1 & 3 & q_k & q_k \\ 3 & 1 & 4 & q_k & q_k \end{pmatrix}$$

Zeilenweise Eingabe der Abschnittslasten
Feld_Nummer, Abschnitts_Nummer, Lastfall_Nummer, F[kN], M[kN*m], W[m], Phi[rad]

$Pkt_LastTragf := \begin{pmatrix} 1 & 2 & 5 & F_{qd} & 0 & 0 & 0 \end{pmatrix}$ $\quad Pkt_LastGebr := \begin{pmatrix} 1 & 2 & 5 & F_{qk} & 0 & 0 & 0 \end{pmatrix}$

**Schnittgrößenermittlung**

Die charakteristischen Einwirkungen wurden bereits mit den hierfür vorgeschriebenen Teilsicherheitsbeiwerten multipliziert. Die maßgebenden Schnittgrößen für die Nachweise im Grenzzustand der Tragfähigkeit sind daher die Minimal- bzw. Maximalwerte, die die DLT-Funktionen ausgeben, wenn als Belastung Ab_LastTragf und Pkt_LastTragf angegeben werden.

Für den Nachweis der Gebrauchstauglichkeit werden die Minimal- bzw. Maximalwerte, die die DLT-Funktionen ausgeben, verwendet, wenn als Belastung Ab_LastGebrauch und Pkt_LastGebrauch angegeben werden. Diese Schnittgrößen werden mit den charakteristischen Einwirkungen berechnet.

## Berechnung der Auflagerkräfte mit Designlasten

AuflKrTragf := DLT_A_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_LastTragf, Pkt_LastTragf, 0.1)

Die Funktion liefert die minimalen und maximalen Auflagerkräfte in einer n*m Matrix.

AuflKrTragf =

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 62.962 | 354.662 |
| 1 | 6 | 107.331 | 355.609 |
| 2 | 11 | 75.524 | 247.102 |
| 3 | 13 | 0 | 0 |

## Berechnung der Momente mit Designlasten

MomTragf := DLT_M_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_LastTragf, Pkt_LastTragf, 1)

Die Funktion liefert die minimalen und maximalen Biegemomente über den Trägerverlauf in einer n*m Matrix.

MomTragf =

| | 0 (m) | 1 (kN · m) | 2 (kN · m) |
|---|---|---|---|
| 0 | 0 | -402.925 | -69.861 |
| 1 | 0.88 | -131.079 | -0.189 |
| 2 | 0.88 | -131.079 | -0.189 |
| 3 | 2.018 | 10.74 | 51.906 |
| 4 | 2.018 | 10.74 | 51.906 |
| 5 | 3.157 | 36.407 | 133.173 |
| 6 | 3.157 | 36.407 | 133.173 |
| 7 | 4.295 | 29.351 | 160.181 |
| 8 | 4.295 | 29.351 | 160.181 |
| 9 | 5.433 | -1.758 | 124.262 |
| 10 | 5.433 | -1.758 | 124.262 |
| 11 | 6.572 | -58.907 | 27.402 |
| 12 | 6.572 | -58.907 | 27.402 |
| 13 | 7.71 | -217.132 | -55.363 |
| 14 | 7.71 | -217.132 | -55.363 |
| 15 | 8.866 | -82.98 | 11.898 |
| 16 | 8.866 | -82.98 | 11.898 |
| 17 | 10.022 | -54.407 | 95.037 |
| 18 | 10.022 | -54.407 | 95.037 |
| 19 | 11.178 | -50.639 | 113.28 |
| 20 | 11.178 | -50.639 | 113.28 |
| 21 | 12.334 | -71.677 | 66.627 |
| 22 | 12.334 | -71.677 | 66.627 |
| 23 | 13.49 | -117.521 | -44.921 |
| 24 | 13.49 | -117.521 | -44.921 |
| 25 | 14.59 | -29.38 | -11.23 |
| 26 | 14.59 | -29.38 | -11.23 |
| 27 | 13 | $1.386 \cdot 10^{-13}$ | $2.984 \cdot 10^{-13}$ |

Zur Bemessing maßgebende Momente in kN/m:

1.Auflager $M_{St1} := 402.925$

Feld 1 $M_{F1} := 160.181$

2.Auflager $M_{St2} := 217.132$

Feld 2 $M_{F2} := 113.28$

3.Auflager $M_{St3} := 117.521$

Kragarm $M_{F3} := 29.38$

## Berechnung der Querkräfte mit Designlasten

QuerkrTragf := DLT_Q_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_LastTragf, Pkt_LastTragf, 1)

Die Funktion liefert die minimalen und maximalen Querkräfte über den Trägerverlauf in einer n*m Matrix.

QuerkrTragf =

| | m | kN | kN |
|---|---|---|---|
| | 0 | 1 | 2 |
| 0 | 0 | 62.962 | 354.662 |
| 1 | 0.88 | 46.627 | 311.927 |
| 2 | 0.88 | 42.509 | 166.045 |
| 3 | 2.018 | 21.379 | 110.765 |
| 4 | 2.018 | 21.379 | 110.765 |
| 5 | 3.157 | 0.249 | 55.485 |
| 6 | 3.157 | 0.249 | 55.485 |
| 7 | 4.295 | -20.882 | 0.204 |
| 8 | 4.295 | -20.882 | 0.204 |
| 9 | 5.433 | -74.909 | -22.179 |
| 10 | 5.433 | -74.909 | -22.179 |
| 11 | 6.572 | -130.189 | -43.31 |
| 12 | 6.572 | -130.189 | -43.31 |
| 13 | 7.71 | -185.47 | -64.44 |
| 14 | 7.71 | 42.892 | 170.14 |
| 15 | 8.866 | 21.433 | 114.002 |
| 16 | 8.866 | 21.433 | 114.002 |
| 17 | 10.022 | -0.025 | 57.863 |
| 18 | 10.022 | -0.025 | 57.863 |
| 19 | 11.178 | -27.988 | 8.23 |
| 20 | 11.178 | -27.988 | 8.23 |
| 21 | 12.334 | -84.126 | -13.228 |
| 22 | 12.334 | -84.126 | -13.228 |
| 23 | 13.49 | -140.265 | -34.687 |
| 24 | 13.49 | 40.838 | 106.838 |
| 25 | 14.59 | 20.419 | 53.419 |
| 26 | 14.59 | 20.419 | 53.419 |
| 27 | 13 | $2.753 \cdot 10^{-14}$ | $7.105 \cdot 10^{-14}$ |

Zur Bemessing maßgebende Querkraft in kN:

1.Auflager $Q_{St1} := 354.662$

2.Auflager links $Q_{St2li} := 185.47$

2.Auflager rechts $Q_{St2re} := 170.14$

3.Auflager links $Q_{St3li} := 140.286$

3.Auflager rechts $Q_{St3re} := 106.838$

## Berechnung der Durchbiegung mit charakteristischen Lasten

DurchbiegGebr := DLT_W_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_LastGebr, Pkt_LastGebr, 1)

Die Funktion liefert die minimale und maximale Durchbiegung über den Trägerverlauf in einer n*m Matrix.

Begrenzung der Verformung nach DIN 1045-1:

$$f \le \frac{l_{eff}}{500}$$ (im Hinblick auf Ausbauten)

DurchbiegGebr =

| | m | m | m |
|---|---|---|---|
| | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.88 | 0 | 0.001 |
| 2 | 0.88 | 0 | 0.001 |
| 3 | 2.018 | 0.001 | 0.003 |
| 4 | 2.018 | 0.001 | 0.003 |
| 5 | 3.157 | 0.001 | 0.004 |
| 6 | 3.157 | 0.001 | 0.004 |
| 7 | 4.295 | 0.001 | 0.005 |
| 8 | 4.295 | 0.001 | 0.005 |
| 9 | 5.433 | 0 | 0.004 |
| 10 | 5.433 | 0 | 0.004 |
| 11 | 6.572 | -0 | 0.002 |
| 12 | 6.572 | -0 | 0.002 |
| 13 | 7.71 | 0 | 0 |
| 14 | 7.71 | 0 | 0 |
| 15 | 8.866 | -0.001 | 0.001 |
| 16 | 8.866 | -0.001 | 0.001 |
| 17 | 10.022 | -0.002 | 0.002 |
| 18 | 10.022 | -0.002 | 0.002 |
| 19 | 11.178 | -0.002 | 0.002 |
| 20 | 11.178 | -0.002 | 0.002 |
| 21 | 12.334 | -0.001 | 0.001 |
| 22 | 12.334 | -0.001 | 0.001 |
| 23 | 13.49 | 0 | 0 |
| 24 | 13.49 | 0 | 0 |
| 25 | 14.59 | -0.001 | 0.002 |
| 26 | 14.59 | -0.001 | 0.002 |
| 27 | 13 | -0.002 | 0.004 |

$$f_1 := 0.005 \quad < \quad \frac{L_1}{500} = 0.015$$

$$f_2 := 0.002 \quad < \quad \frac{L_2}{500} = 0.012$$

$$f_3 := 0.004 \quad = \quad \frac{L_3}{500} = 0.004$$

**Biegebemessung des Stahlbetonquerschnittes**

Beton_Nachweise-Funktion: BBem

BBem(Breite [m], Hoehe [m],e_Bewehrung [m], Msd [KNm], Nsd [KN], fcd [MN / m^2])

Bemessung von Betonquerschnitten nach DIN 1045-1 mit dem dimensionslosen OMEGA-Verfahren Die Bemessung beschränkt sich auf Rechteckquerschnitte. Es können Biegemomente und Normalkräfte Berücksichtigt werden.

Rückgabewert ist die erforderliche Biegebewehrung auf der Zug- und auf der Druckseite in $cm^2$.

Betondeckung in m $c := 0.035$

Bemessungs-Betondruckfestigkeit in MN/m²

Vorwerte: für Normalbeton $\alpha := 0.85$ $\gamma_c := 1.5$

Teilsicherheitsbeiwert für Beton

$$f_{cd} := \alpha \cdot \frac{f_{ck}}{\gamma_c}$$ $f_{cd} = 19.833$

**Einspannung**

$as_{St1} := BBem(b, h, c, M_{St1}, 0, f_{cd})$

Erforderliche Biegebewehrung in cm² unten: $as_{St1_{0,1}} = 1.182$

oben: $as_{St1_{0,0}} = 23.036$

$$as_{St1}^T = \begin{pmatrix} 23.036 \\ 1.182 \end{pmatrix}$$

**Feld 1**

$as_{F1} := BBem(b, h, c, M_{F1}, 0, f_{cd})$

Erforderliche Biegebewehrung in cm² oben: $as_{F1_{0,1}} = 0$

unten: $as_{F1_{0,0}} = 8.565$

**1. Zwischenauflager**

$as_{St2} := BBem(b, h, c, M_{St2}, 0, f_{cd})$

Erforderliche Biegebewehrung in cm² unten: $as_{St2_{0,1}} = 0$

oben: $as_{St2_{0,0}} = 11.796$

**Feld 2**

$as_{F2} := BBem(b, h, c, M_{F2}, 0, f_{cd})$

Erforderliche Biegebewehrung in cm² oben: $as_{F2_{0,1}} = 0$

unten: $as_{F2_{0,0}} = 5.979$

**2. Zwischenauflager**

$as_{St3} := BBem(b, h, c, M_{St3}, 0, f_{cd})$

Erforderliche Biegebewehrung in cm² unten: $as_{St3_{0,1}} = 0$

oben: $as_{St3_{0,0}} = 6.207$

**Kragarm**

$as_{F3} := BBem(b, h, c, M_{F3}, 0, f_{cd})$

Erforderliche Biegebewehrung in cm² unten: $as_{F3_{0,1}} = 0$

oben: $as_{F3_{0,0}} = 1.546$

**Querkraftbemessung des Stahlbetonquerschnittes**

Beton_Nachweise-Funktion: NormalB_Bst500_BVertikaleBuegel

NormalB_Bst500_VertikaleBuegel(V_sd [kN], d [m], bw [m], f_ck [kN/m^2], A_sl[cm^2], sigma_cd [MN/m^2],z [m])

Die Querkraftbemessung von Stahlbeton-Rechteckquerschnitten nach DIN 1045-1. Die Funktion liefert den erforderlichen Gesamtquerschnitt an Schubbewehrung in [cm^2] für Normalbeton mit senkrechter Bügelbewehrung der Betonstahlsorte BSt 500

Hebelarm der inneren Kräfte in m $d := 0.9 \cdot (h - c)$ $d = 0.418$

**Einspannung**

-eingelegte Längsbewehrung $as1 := as_{St1_{0,1}} + as_{St1_{0,0}}$

$abü_{St1} := NormalB_Bst500_VertikaleBuegel(Q_{St1}, h, b, f_{cd}, as1, 0, d)$ $as1 = 24.218$

Erforderliche Bügelbewehrung in cm² $abü_{St1} = 12.508$

## 1. Zwischenauflager

-eingelegte Längsbewehrung $as2 := as_{St2_{0,1}} + as_{St2_{0,0}}$

$$abü_{St2li} := NormalB_Bst500_VertikaleBuegel\left(Q_{St2li}, h, b, f_{cd}, as2, 0, d\right)$$

$$abü_{St2re} := NormalB_Bst500_VertikaleBuegel\left(Q_{St2re}, h, b, f_{cd}, as2, 0, d\right)$$

Erforderliche Bügelbewehrung in cm² links $abü_{St2li} = 4.759$

rechts $abü_{St2re} = 4.057$

## 2. Zwischenauflager

-eingelegte Längsbewehrung $as3 := as_{St3_{0,1}} + as_{St3_{0,0}}$

$$abü_{St3li} := NormalB_Bst500_VertikaleBuegel\left(Q_{St3li}, h, b, f_{cd}, as3, 0, d\right)$$

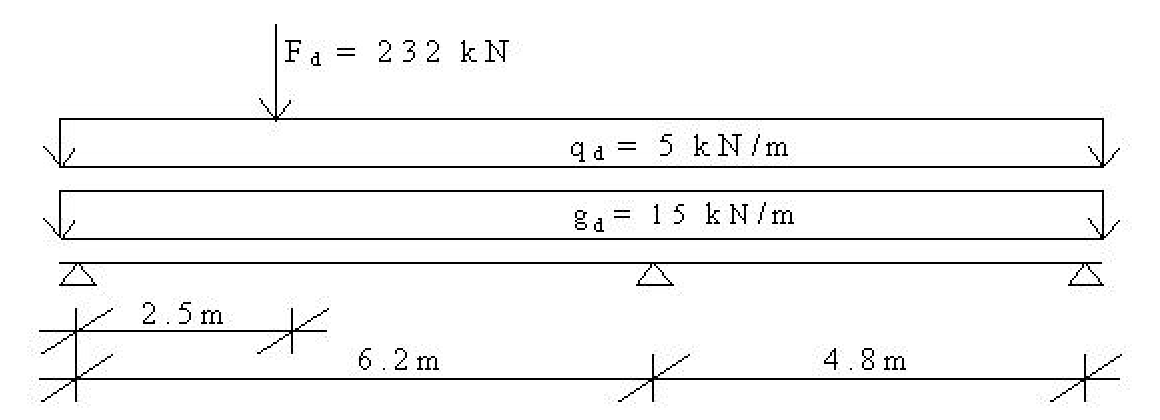
$$abü_{St3re} := NormalB_Bst500_VertikaleBuegel\left(Q_{St3re}, h, b, f_{cd}, as3, 0, d\right)$$

Erforderliche Bügelbewehrung in cm² links $abü_{St3li} = 2.689$

rechts $abü_{St3re} = 1.957$

## 5.2 Zweifeldträger in Stahlbeton

### Systemskizze:




Querschnittswerte: Breite in m $b := 0.26$ Höhe in m $h := 0.48$

Trägheitsmodul $I := b \cdot \frac{h^3}{12}$ $I = 2.396 \times 10^{-3}$

Material: - Beton C35/45 - Betonstahl BSt 500 S $m^4$

Druckfestigeit in N/mm² $f_{ck} := 35$ Nennstreckgrenze in N/mm² $f_{yk} := 500$

Elastizitätsmodul in kN/m² $E := 33300000$

$E \cdot I = 7.979 \times 10^4$ kN/m²

Feldlängen [m] Abschnittslängen [m]

$L_1 := 6.2$ $l_{a1} := 2.5$

$L_2 := 4.8$ $l_{a2} := 3.7$

$l_{a3} := 4.8$

Belastung $F_d := 232$ $q_d := 5$ $g_d := 15$

## Eingabewerte für die DLT Funktionen

Eingabe der Abschnitte: (Feld_Nummer, Abschnitts_Nummer, Abschnitts_Länge[m])

$$Ab := \begin{pmatrix} 1 & 1 & l_{a1} \\ 1 & 2 & l_{a2} \\ 2 & 1 & l_{a3} \end{pmatrix}$$

Eingabe der Querschnittsdefinitionen [kN*m^2]

$$EI := (E \cdot I \quad E \cdot I)$$

Eingabe des Bettungsmodul [kN/m^3]

$$Cs := (0 \quad 0)$$

Eingabe der Normalkraft Th.II Ordnung [kN]: N = 0 ( Th. I Ordnung )

$$N := (0 \quad 0)$$

Lagerdefinition am linken Trägerrand

$$Lli := 1$$

Lagerdefinition am rechten Trägerrand

$$Lre := 1$$

Eingabe der Abschnittslasten: Feld_Nr, Abschnitts_Nr, LF_Nr, q_li [kN/m], q_re [kN/m]

$$Ab_Last := \begin{pmatrix} 1 & 1 & 1 & g_d & g_d \\ 1 & 2 & 1 & g_d & g_d \\ 2 & 1 & 1 & g_d & g_d \\ 1 & 1 & 2 & q_d & q_d \\ 1 & 2 & 2 & q_d & q_d \\ 2 & 1 & 3 & q_d & q_d \end{pmatrix}$$

Eingabe der Abschnittslasten: Feld_Nr, Abschnitts_Nr, LF_Nr, F[kN], M[kN*m], W[m], Phi[rad]

$$Pkt_Last := \begin{pmatrix} 1 & 2 & 4 & F_d & 0 & 0 & 0 \end{pmatrix}$$

## Schnittgrößenermittlung

## Berechnung der Auflagerkräfte

$$AuflKr := DLT_A_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1)$$

AuflKr =

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 35.894 | 166.598 |
| 1 | 5 | 104.483 | 283.453 |
| 2 | 9 | -7.728 | 34.3 |

## Berechnung der Momente

Mom := DLT_M_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1.5)

Mom =

| | 0 (m) | 1 (kN · m) | 2 (kN · m) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 2.5 | 42.859 | 353.994 |
| 2 | 2.5 | 42.859 | 353.994 |
| 3 | 4.35 | 14.219 | 106.275 |
| 4 | 4.35 | 14.219 | 106.275 |
| 5 | 6.2 | -216.178 | -59.475 |
| 6 | 6.2 | -216.178 | -59.475 |
| 7 | 7.8 | -101.53 | 7.361 |
| 8 | 7.8 | -101.53 | 7.361 |
| 9 | 9.4 | -31.565 | 29.28 |
| 10 | 9.4 | -31.565 | 29.28 |
| 11 | 9 | -0 | -0 |

Zur Bemessing maßgebende Momente in kN/m:

Feld 1 (positiv) $M_{F1} := 353.994$

Zwischenauflager (negativ) $M_{St} := 216.178$

Feld 2 (positiv) $M_{F2} := 29.28$

## Berechnung der Querkräfte

Querkr := DLT_Q_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1.5)

Querkr =

| | 0 (m) | 1 (kN) | 2 (kN) |
|---|---|---|---|
| 0 | 0 | 35.894 | 166.598 |
| 1 | 2.5 | -1.606 | 116.598 |
| 2 | 2.5 | -117.232 | 0.223 |
| 3 | 4.35 | -153.416 | -28.343 |
| 4 | 4.35 | -153.416 | -28.343 |
| 5 | 6.2 | -190.416 | -56.093 |
| 6 | 6.2 | 48.391 | 93.037 |
| 7 | 7.8 | 24.391 | 61.037 |
| 8 | 7.8 | 24.391 | 61.037 |
| 9 | 9.4 | -2.3 | 31.728 |
| 10 | 9.4 | -2.3 | 31.728 |
| 11 | 9 | -34.3 | 7.728 |

Zur Bemessing maßgebende Querkraft in kN:

1.Auflager $Q_{St1} := 166.598$

2.Auflager links $Q_{St2li} := 190.416$

2.Auflager rechts $Q_{St2re} := 93.037$

3.Auflager $Q_{St3} := 34.3$

## Berechnung der Durchbiegung

Durchbieg := DLT_W_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1)

Begrenzung der Verformung nach DIN 1045-1:

$f \leq \frac{l_{eff}}{500}$ (im Hinblick auf Ausbauten)

Durchbieg =

| | 0 (m) | 1 (m) | 2 (m) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1.25 | 0.001 | 0.008 |
| 2 | 1.25 | 0.001 | 0.008 |
| 3 | 2.5 | 0.002 | 0.012 |
| 4 | 2.5 | 0.002 | 0.012 |
| 5 | 3.733 | 0.001 | 0.011 |
| 6 | 3.733 | 0.001 | 0.011 |
| 7 | 4.967 | 0.001 | 0.005 |
| 8 | 4.967 | 0.001 | 0.005 |
| 9 | 6.2 | 0 | 0 |
| 10 | 6.2 | 0 | 0 |
| 11 | 7.4 | -0.002 | 0 |
| 12 | 7.4 | -0.002 | 0 |
| 13 | 8.6 | -0.002 | 0.001 |
| 14 | 8.6 | -0.002 | 0.001 |
| 15 | 9.8 | -0.001 | 0 |
| 16 | 9.8 | -0.001 | 0 |
| 17 | 9 | 0 | 0 |

$f_1 := 0.012 \quad = \quad \frac{L_1}{500} = 0.012$

$f_2 := 0.002 \quad < \quad \frac{L_2}{500} = 0.01$

## Biegebemessung des Stahlbetonquerschnitts

Beton_Nachweise-Funktion: BBem

BBem(Breite [m], Hoehe [m],e_Bewehrung [m], Msd [KNm], Nsd [KN], fcd [MN / m^2])

Betondeckung in m $c := 0.035$

Bemessungs-Betondruckfestigkeit in MN/m²

Vorwerte: für Normalbeton $\alpha := 0.85$

Teilsicherheitsbeiwert für Beton $\gamma_c := 1.5$

$f_{cd} := \alpha \cdot \frac{f_{ck}}{\gamma_c}$ $f_{cd} = 19.833$

## Feld 1

$as_{F1} := BBem(b, h, c, M_{F1}, 0, f_{cd})$

Erforderliche Biegebewehrung in cm² oben: $as_{F1_{0,1}} = 2.901$

unten: $as_{F1_{0,0}} = 21.026$

## 1. Zwischenauflager

$as_{St} := BBem(b,h,c,M_{St},0,f_{cd})$

Erforderliche Biegebewehrung in cm² unten: $as_{St_{0,1}} = 0$

oben: $as_{St_{0,0}} = 12.506$

## Feld 2

$as_{F2} := BBem(b,h,c,M_{F2},0,f_{cd})$

Erforderliche Biegebewehrung in cm² oben: $as_{F2_{0,1}} = 0$

unten: $as_{F2_{0,0}} = 1.66$

## Querkraftbemessung des Stahlbetonquerschnittes

Beton_Nachweise-Funktion: NormalB_Bst500_VertikaleBuegel

NormalB_Bst500_VertikaleBuegel(V_sd [kN], d [m], bw [m], f_ck [kN/m^2], A_sl [cm^2], sigma_cd [MN/m^2], z[m])

Hebelarm der inneren Kräfte in m $d := 0.9 \cdot (h - c)$ $d = 0.4$

## Auflager links

-eingelegte Längsbewehrung $as1 := 0$

$abü_{St1} := NormalB_Bst500_VertikaleBuegel(Q_{St1},h,b,f_{cd},as1,0,d)$

Erforderliche Bügelbewehrung in cm² $abü_{St1} = 4.735$

## Zwischenauflager

-eingelegte Längsbewehrung $as2 := as_{St_{0,1}} + as_{St_{0,0}}$

$abü_{Stli} := NormalB_Bst500_VertikaleBuegel(Q_{St2li},h,b,f_{cd},as2,0,d)$

$abü_{Stre} := NormalB_Bst500_VertikaleBuegel(Q_{St2re},h,b,f_{cd},as2,0,d)$

Erforderliche Bügelbewehrung in cm² links $abü_{Stli} = 5.875$

rechts $abü_{Stre} = 1.781$

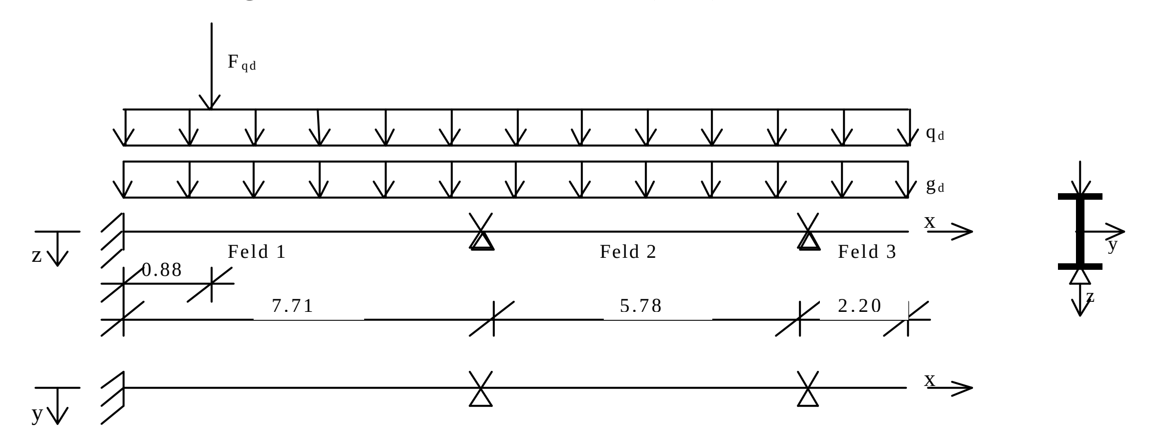
## Auflager rechts:

-eingelegte Längsbewehrung $as3 := 0$

$abü_{St3} := NormalB_Bst500_VertikaleBuegel(Q_{St3},h,b,f_{cd},as3,0,d)$

Erforderliche Bügelbewehrung in cm² $abü_{St3} = 1.97$

## 5.3 Durchlaufträger in Stahl nach DIN 18 800 (3.91)




### Systemwerte:

Feldlängen [m] — Abschnittslängen [m]

$L_1 := 7.71$

$L_2 := 5.78$

$L_3 := 2.20$

$l_{a1} := 0.88$

$l_{a2} := 6.83$

$l_{a3} := 5.78$

$l_{a4} := 2.20$

Querschnittswerte: IPE 450

Trägheitsmodul in m^4 $I_1 := 3.374 \cdot 10^{-4}$

Trägheitsmodul 2.Ordnung in m^3 $W_1 := 15.0 \cdot 10^{-4}$

Eigengewicht in kN/m $g_1 := 0.776$

Schubfläche in m² $A_{Steg1} := 3.55 \cdot 10^{-3}$

Gesamtläche in m² $A_1 := 9.88 \cdot 10^{-3}$

Material: St 52 Elastizitätsmodul in kN/m² $E := 2.1 \cdot 10^8$

Streckgrenze in kN/m² $f_{yk} := 36$

Teilsicherheitswert der Widerstandsgrößen $\gamma_M := 1.1$

Elastizitätsmodul zur Berechnung der Steifigkeit in kN/m² $E_1 := \frac{E}{\gamma_M}$

$E_1 = 1.909 \times 10^8$

Grenznormalspannung in kN/m² $\sigma_{Rd} := \frac{f_{yk}}{\gamma_M}$ $\sigma_{Rd} = 32.727$

Grenzschubspannung in kN/m² $\tau_{Rd} := \frac{f_{yk}}{\gamma_M \cdot \sqrt{3}}$ $\tau_{Rd} = 18.895$

**Belastungszusammenstellung:**

Teilsicherheitsbeiwerte der Einwirkungen $\gamma_G := 1.35$ $\gamma_Q := 1.50$

Ständige Lasten:

- ständigeLasten in kN/m, die nicht aus dem Eigengewicht stammen $g_s := 10$

Summe der ständigen Lasten in kN/m $g_{k1} := g_1 + g_s$ $g_{k1} = 10.776$

Verkehrslast: Streckenlast in kN/m $q_k := 20$ Einzellast in kN $F_{qk} := 100$

Belastung auf Designniveau:

$g_{d1} := \gamma_G \cdot g_{k1}$ $q_d := \gamma_Q \cdot q_k$ $F_{qd} := \gamma_Q \cdot F_{qk}$

$g_{d1} = 14.548$ $q_d = 30$ $F_{qd} = 150$

## Eingabewerte für die DLT Funktionen

Zeilenweise Eingabe der Abschnitte
(Feld_Nummer, Abschnitts_Nummer, Abschnitts_Länge[m]

$$Ab := \begin{pmatrix} 1 & 1 & l_{a1} \\ 1 & 2 & l_{a2} \\ 2 & 1 & l_{a3} \\ 3 & 1 & l_{a4} \end{pmatrix}$$

Feldweise Eingabe der Querschnittsdefinitionen [kN*m^2]

$$EI := \begin{pmatrix} E_1 \cdot I_1 & E_1 \cdot I_1 & E_1 \cdot I_1 \end{pmatrix}$$

Feldweise Eingabe des Bettungsmodul [kN/m^3]

$$Cs := \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

Feldweise Eingabe der Normalkraft Th.II Ordnung [kN]
N = 0 ( Th. I Ordnung )

$$N := \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

Lagerdefinition am linken Trägerrand

$$Lli := 2$$

Lagerdefinition am rechten Trägerrand

$$Lre := 3$$

Zeilenweise Eingabe der Abschnittslasten
Feld_Nummer, Abschnitts_Nummer, Lastfall_Nummer, q_li [kN/m], q_re [kN/m]

$$Ab_LastTragf := \begin{pmatrix} 1 & 1 & 1 & g_{d1} & g_{d1} \\ 1 & 2 & 1 & g_{d1} & g_{d1} \\ 2 & 1 & 1 & g_{d1} & g_{d1} \\ 3 & 1 & 1 & g_{d1} & g_{d1} \\ 1 & 1 & 2 & q_d & q_d \\ 1 & 2 & 2 & q_d & q_d \\ 2 & 1 & 3 & q_d & q_d \\ 3 & 1 & 4 & q_d & q_d \end{pmatrix} \qquad Ab_LastGebr := \begin{pmatrix} 1 & 1 & 1 & g_{k1} & g_{k1} \\ 1 & 2 & 1 & g_{k1} & g_{k1} \\ 2 & 1 & 1 & g_{k1} & g_{k1} \\ 3 & 1 & 1 & g_{k1} & g_{k1} \\ 1 & 1 & 2 & q_k & q_k \\ 1 & 2 & 2 & q_k & q_k \\ 2 & 1 & 3 & q_k & q_k \\ 3 & 1 & 4 & q_k & q_k \end{pmatrix}$$

Zeilenweise Eingabe der Abschnittslasten
Feld_Nummer, Abschnitts_Nummer, Lastfall_Nummer, F[kN], M[kN*m], W[m], Phi[rad]

$Pkt_LastTragf := \begin{pmatrix} 1 & 2 & 5 & F_{qd} & 0 & 0 & 0 \end{pmatrix}$ $Pkt_LastGebr := \begin{pmatrix} 1 & 2 & 5 & F_{qk} & 0 & 0 & 0 \end{pmatrix}$

**Schnittgrößenermittlung**

Die maßgebenden Schnittgrößen für die Nachweise im Grenzzustand der Tragfähigkeit, sind die Minimal- bzw. Maximalwerte, die die DLT-Funktionen ausgeben, wenn als Belastung Ab_LastTragf und Pkt_LastTragf angegeben werden, da die charakteristischen Einwirkungen bereits mit den hierfür vorgeschriebenen Teilsicherheitsbeiwerten multipliziert wurden.

Für den Nachweis der Gebrauchstauglichkeit werden die Minimal- bzw. Maximalwerte, die die DLT-Funktionen ausgeben, verwendet, wenn als Belastung Ab_LastGebrauch und Pkt_LastGebrauch angegeben werden. Diese Schnittgrößen werden mit den charakteristischen Einwirkungen berechnet.

**Berechnung der Auflagerkräfte mit Designlasten**

AuflKrTragf := DLT_A_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_LastTragf, Pkt_LastTragf, 1)

Die Funktion liefert die minimalen und maximalen Auflagerkräfte in einer n*m Matrix.

AuflKrTragf =

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 46.709 | 338.408 |
| 1 | 6 | 79.957 | 328.235 |
| 2 | 11 | 56.158 | 227.736 |
| 3 | 13 | 0 | 0 |

**Berechnung der Momente mit Designlasten:**

MomTragf := DLT_M_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_LastTragf, Pkt_LastTragf, 1)

Die Funktion liefert die minimalen und maximalen Biegemomente über den Trägerverlauf in einer n*m Matrix.

MomTragf := DLT_M_MinMax(Ab , EI , Cs , N , Lli , Lre , Ab_LastTragf , Pkt_LastTragf , 1)

Die Funktion liefert die minimalen und maximalen Biegemomente über den Trägerverlauf in einer n*m Matrix.

MomTragf =

| | m | kN · m | kN · m |
|---|---|---|---|
| | 0 | 1 | 2 |
| 0 | 0 | -381.042 | -47.978 |
| 1 | 0.88 | -121.945 | 8.946 |
| 2 | 0.88 | -121.945 | 8.946 |
| 3 | 2.018 | 7.996 | 49.162 |
| 4 | 2.018 | 7.996 | 49.162 |
| 5 | 3.157 | 26.987 | 123.752 |
| 6 | 3.157 | 26.987 | 123.752 |
| 7 | 4.295 | 18.457 | 149.286 |
| 8 | 4.295 | 18.457 | 149.286 |
| 9 | 5.433 | -8.924 | 117.096 |
| 10 | 5.433 | -8.924 | 117.096 |
| 11 | 6.572 | -57.142 | 29.168 |
| 12 | 6.572 | -57.142 | 29.168 |
| 13 | 7.71 | -201.233 | -39.464 |
| 14 | 7.71 | -201.233 | -39.464 |
| 15 | 8.866 | -79.048 | 15.831 |
| 16 | 8.866 | -79.048 | 15.831 |
| 17 | 10.022 | -57.076 | 92.368 |
| 18 | 10.022 | -57.076 | 92.368 |
| 19 | 11.178 | -54.545 | 109.374 |
| 20 | 11.178 | -54.545 | 109.374 |
| 21 | 12.334 | -71.455 | 66.85 |
| 22 | 12.334 | -71.455 | 66.85 |
| 23 | 13.49 | -107.805 | -35.205 |
| 24 | 13.49 | -107.805 | -35.205 |
| 25 | 14.59 | -26.951 | -8.801 |
| 26 | 14.59 | -26.951 | -8.801 |
| 27 | 13 | $-1.279 \cdot 10^{-13}$ | $-5.507 \cdot 10^{-14}$ |

Zur Bemessing maßgebende Momente in kN/m:

1.Auflager $M_{St1} := 381.042$

Feld 1 $M_{F1} := 149.286$

2.Auflager $M_{St2} := 201.233$

Feld 2 $M_{F2} := 109.374$

3.Auflager $M_{St3} := 107.805$

Kragarm $M_{F3} := 107.805$

## Berechnung der Querkräfte mit Designlasten

QuerkrTragf := DLT_Q_MinMax(Ab , EI , Cs , N , Lli , Lre , Ab_LastTragf , Pkt_LastTragf , 1)

Die Funktion liefert die minimalen und maximalen Querkräfte über den Trägerverlauf in einer n*m Matrix.

QuerkrTragf =

| | m | kN | kN |
|---|---|---|---|
| | 0 | 1 | 2 |
| 0 | 0 | 46.709 | 338.408 |
| 1 | 0.88 | 33.907 | 299.207 |
| 2 | 0.88 | 29.789 | 153.325 |
| 3 | 2.018 | 13.229 | 102.615 |
| 4 | 2.018 | 13.229 | 102.615 |
| 5 | 3.157 | -3.331 | 51.905 |
| 6 | 3.157 | -3.331 | 51.905 |
| 7 | 4.295 | -19.891 | 1.195 |
| 8 | 4.295 | -19.891 | 1.195 |
| 9 | 5.433 | -69.348 | -16.618 |
| 10 | 5.433 | -69.348 | -16.618 |
| 11 | 6.572 | -120.058 | -33.178 |
| 12 | 6.572 | -120.058 | -33.178 |
| 13 | 7.71 | -170.768 | -49.738 |
| 14 | 7.71 | 30.219 | 157.467 |
| 15 | 8.866 | 13.402 | 105.97 |
| 16 | 8.866 | 13.402 | 105.97 |
| 17 | 10.022 | -3.415 | 54.473 |
| 18 | 10.022 | -3.415 | 54.473 |
| 19 | 11.178 | -26.737 | 9.481 |
| 20 | 11.178 | -26.737 | 9.481 |
| 21 | 12.334 | -78.234 | -7.336 |
| 22 | 12.334 | -78.234 | -7.336 |
| 23 | 13.49 | -129.731 | -24.153 |
| 24 | 13.49 | 32.005 | 98.005 |
| 25 | 14.59 | 16.002 | 49.002 |
| 26 | 14.59 | 16.002 | 49.002 |
| 27 | 13 | $-2.842\cdot10^{-14}$ | $-1.399\cdot10^{-14}$ |

Zur Bemessing maßgebende Querkraft in kN:

1.Auflager $Q_{St1} := 338.408$

2.Auflager links $Q_{St2li} := 170.768$

2.Auflager rechts $Q_{St2re} := 157.467$

3.Auflager links $Q_{St3li} := 129.731$

3.Auflager rechts $Q_{St3re} := 98.005$

**Berechnung der Durchbiegung mit charakteristischen Lasten:**

DurchbiegGebr := DLT_W_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_LastGebr, Pkt_LastGebr, 1)

Die Funktion liefert die minimale und maximale Durchbiegung über den Trägerverlauf in einer n*m Matrix.

$E \cdot I_1 = 70854$

DurchbiegGebr =

| | m | m | m |
|---|---|---|---|
| | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.88 | 0 | 0.001 |
| 2 | 0.88 | 0 | 0.001 |
| 3 | 2.018 | 0.001 | 0.004 |
| 4 | 2.018 | 0.001 | 0.004 |
| 5 | 3.157 | 0.001 | 0.006 |
| 6 | 3.157 | 0.001 | 0.006 |
| 7 | 4.295 | 0.001 | 0.007 |
| 8 | 4.295 | 0.001 | 0.007 |
| 9 | 5.433 | 0 | 0.006 |
| 10 | 5.433 | 0 | 0.006 |
| 11 | 6.572 | -0 | 0.003 |
| 12 | 6.572 | -0 | 0.003 |
| 13 | 7.71 | 0 | 0 |
| 14 | 7.71 | 0 | 0 |
| 15 | 8.866 | -0.002 | 0.002 |
| 16 | 8.866 | -0.002 | 0.002 |
| 17 | 10.022 | -0.003 | 0.003 |
| 18 | 10.022 | -0.003 | 0.003 |
| 19 | 11.178 | -0.003 | 0.003 |
| 20 | 11.178 | -0.003 | 0.003 |
| 21 | 12.334 | -0.002 | 0.002 |
| 22 | 12.334 | -0.002 | 0.002 |
| 23 | 13.49 | 0 | 0 |
| 24 | 13.49 | 0 | 0 |
| 25 | 14.59 | -0.002 | 0.003 |
| 26 | 14.59 | -0.002 | 0.003 |
| 27 | 13 | -0.003 | 0.006 |

Begrenzung der Verformung

allg.: $f \le \frac{l}{250}$

$f_1 := 0.007 \quad < \quad \frac{L_1}{250} = 0.031$

$f_2 := 0.003 \quad < \quad \frac{L_2}{250} = 0.023$

Kragarm.: $f_k \le \frac{l}{200}$

$f_3 := 0.006 \quad < \quad \frac{L_3}{200} = 0.011$

**Schubnachweis der Querschnitte:**

Vereinfacht $\tau_d = Q_d / A_{Steg}$

$\tau_{Rd} = 18.895$

- maximale Querkraft: $Q_{St1}$

$\frac{Q_{St1} \cdot 10^{-4}}{A_{Steg1}} = 9.533 \quad < \quad \tau_{Rd}$

**Eingabewerte für die Stahlbau-Funktionen:**

Sigma_Nachweis_My(Nd, Myd, Vzd, Profil, Stahlguete, Begrenzung)

Die Funktion führt den allgemeinen Spannungsnachweis für einachsige Biegung um die Y-Achse mit Normalkraft. Rückgabewert ist der Spannungsnachweis Sd/Rd.

Nachweis im Bereich der Einspannung (maßgebend für den Spannungsnachweis des gesamten Trägers, da hier sowohl die höchste Querkraft, als auch das höchste Moment auftreten. Der Durchlaufträger ist frei von Normalkräften.)

$$\text{Sigma_Nachweis_My}\left[0, M_{St1}, Q_{St1}, (2\ \ 450), 4, 1\right] = 0.759 \qquad < 1$$

**Eingabewerte für die BDK-Funktionen**

Davon ausgehend, dass an allen Auflagern die Bedingungen, für das Vorhandensein einer Gabellagerung, erfüllt sind, muss für Feld 1 und Feld 2 ein Biegeknicknachweis und ein Biegedrillknicknachweis geführt werden. Für den Kragarm (Feld 3) wird nur ein Biegeknicknachweis geführt.

Im Rahmen dieser Arbeit werden anzusetzende Vorverformungen nicht berücksichtigt.

Funktion:
biegungEinachsig(Einwirkung[N|My],Profil,Stahlguete,l[m],betaY|Z,betaY_knicken,betaY_drill,zp[m],Xi)

Nachweis für Feld 1:

- maßgebendes Moment: $M_{St1}$ in Auflagerbereich, $M_{F1}$ im Feldbereich
- Profil: IPE 450
- Stahlgüte: St 52
- Länge: 7.71m

- Knicklängenbeiwerte in beiden Richtungen (dieser Wert liegt auf der sicheren Seite, da durch die Teileinspannung am rechten Ende dieses Feldes Knicklängenbeiwert kleiner wird) $\beta := 0.7$

- Ermittlung des Momentenbeiwertes für Biegeknicken:

$$\psi := \frac{M_{St2}}{M_{St1}} \qquad \psi = 0.528 \quad < 0.77\ : \qquad \beta_{mBK} := 1$$

- Ermittlung des Momentenbeiwertes für Biegedrillknicken

$$\beta_{M\psi} := 1.8 - 0.7 \cdot \psi \qquad \beta_{M\psi} = 1.43$$

$$M_Q := \frac{\left(|M_{St1}| + |M_{St2}|\right)}{2} + |M_{F1}| \quad \text{(näherungsweise)} \qquad M_Q = 440.423$$

$$\Delta M := |M_{St1}| + |M_{F1}| \qquad \Delta M = 530.328$$

$\beta_{MQ} := 1.3$

$$\beta_{mBDK} := \beta_{M\psi} + \left(\frac{M_Q}{\Delta M}\right) \cdot \left(\beta_{MQ} - \beta_{M\psi}\right)$$

$\beta_{mBDK} = 1.322$

- $zp_z := 0.225$ wenn die Querkraft am Zuggurt angreift (im Bereich der Auflager)
- $zp_z := -0.225$ wenn die Querkraft am Druckgurt angreift (im Feldbereich)
- Momentenbeiwert für die Gabellagerung $Xi := 1.12$

Nachweis im Bereich des linken Lagers:

$$\text{biegungEinachsig}\left[\left(0 \quad M_{St1}\right), \left(2 \quad 450\right), 4, L_1, \left(\beta \quad \beta\right), \beta_{mBK}, \beta_{mBDK}, zp_z, Xi\right] = \left(0.679 \quad 0.921\right)$$

| BK | BDK |
|---|---|
| 0.679 | 0.921 |
| <1 | <1 |

Nachweis im Bereich des Feldes:

$$\text{biegungEinachsig}\left[\left(0 \quad M_{F1}\right), \left(2 \quad 450\right), 4, L_1, \left(\beta \quad \beta\right), \beta_{mBK}, \beta_{mBDK}, zp_D, Xi\right] = \left(0.266 \quad 0.612\right)$$

| BK | BDK |
|---|---|
| 0.266 | 0.612 |
| <1 | <1 |

Nachweis für Feld 2:

- maßgebendes Moment: $M_{St2}$ für Auflagerbereich, $M_{F2}$ für Feldbereich
- Profil: IPE 450
- Stahlgüte: St 52
- Länge: 5.78m

- Ermittlung des Momentenbeiwertes für Biegeknicken:

$$\psi := \frac{M_{St3}}{M_{St2}} \qquad \psi = 0.536 < 0.77 :$$

$\beta_{mBK} := 1.0$

- Ermittlung des Momentenbeiwertes für Biegedrillknicken

$\beta_{M\psi} := 1.8 - 0.7 \cdot \psi$ $\qquad \beta_{M\psi} = 1.425$

$$M_Q := \frac{\left(\left|M_{St2}\right| + \left|M_{St3}\right|\right)}{2} + \left|M_{F2}\right| \quad \text{(näherungsweise)} \qquad M_Q = 263.893$$

$$\Delta M := \left|M_{St2}\right| + \left|M_{F2}\right| \qquad \Delta M = 310.607$$

$\beta_{MQ} := 1.3$

$$\beta_{mBDK} := \beta_{M\psi} + \left(\frac{M_Q}{\Delta M}\right) \cdot \left(\beta_{MQ} - \beta_{M\psi}\right)$$

$\beta_{mBDK} = 1.319$

- $zp_z := 0.225$ wenn die Querkraft am Zuggurt angreift (im Bereich der Auflager)
- $zp_z := -0.225$ wenn die Querkraft am Druckgurt angreift (im Feldbereich)
- Momentenbeiwert für die Gabellagerung Xi := 1.12

Nachweis im Bereich des ersten Zwischenlagers:

$$\text{biegungEinachsig}\left[\left(0 \;\; M_{St2}\right),\left(2 \;\; 450\right),4,L_2,\left(\beta \;\; \beta\right),\beta_{mBK},\beta_{mBDK},zp_z,Xi\right] = \begin{matrix} BK & BDK \\ (0.358 & 0.592) \\ \le 1 & \le 1 \end{matrix}$$

Nachweis im Bereich des Feldes:

$$\text{biegungEinachsig}\left[\left(0 \;\; M_{F2}\right),\left(2 \;\; 450\right),4,L_2,\left(\beta \;\; \beta\right),\beta_{mBK},\beta_{mBDK},zp_D,Xi\right] = \begin{matrix} BK & BDK \\ (0.195 & 0.739) \\ \le 1 & \le 1 \end{matrix}$$

Nachweis für Feld 3:

- maßgebendes Moment: $M_{St3}$
- Profil: IPE 450
- Stahlgüte: St 52
- Länge: 2.20m

- Knicklängenbeiwerte in beiden Richtungen $\beta := 3.5$
  ( Ermittelt über die Drehfeder, die durch Feld 2 vorhanden ist)

- Ermittlung des Momentenbeiwertes für Biegeknicken:

$\psi := 0$ $\beta_{mBK} := 1.0$

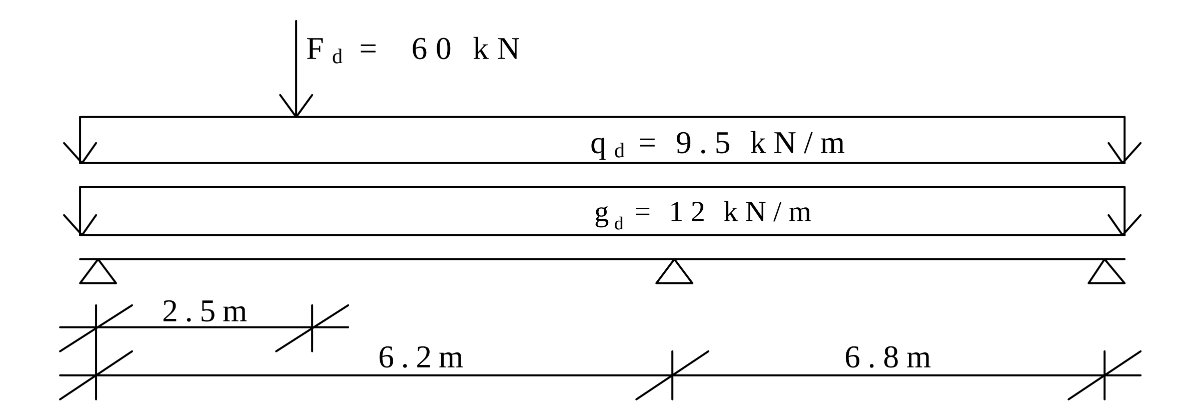
- Ermittlung des Momentenbeiwertes für Biegedrillknicken
  Da am Ende des Kragarms keine Gabellagerung vorhanden ist, kann kein Biegedrillknicken stattfinden. Deswegen wird der Momentenbeiwerte für Biegedrillknicken gleich Null gesetzt.

$\beta_{mBDK} := 0$

- $zp_z := 0.225$ wenn die Querkraft am Zuggurt angreift (im Bereich der Auflager)
- $zp_D := -0.225$ wenn die Querkraft am Druckgurt angreift (im Feldbereich)

## 5.4 Zweifeldträger in Stahl

### Systemskizze




### System- und Querschnittswerte

IPE 450

Trägheitsmodul in m^4 $I = 3.374 \times 10^{-4}$

Schubfläche in m² $A_{Steg} = 0.004$

Gesamtläche in m² $A = 0.01$

Material: St 52 Elastizitätsmodul in kN/m² $E := 2.1 \cdot 10^{8}$

Feldlängen [m]

$L_1 := 6.2$

$L_2 := 6.8$

Abschnittslängen [m]

$l_{a1} := 2.5$

$l_{a2} := 3.7$

$l_{a3} := 6.8$

Belastung $F_d := 60$ $q_d := 9.5$ $g_d := 12$

## Eingabewerte für die DLT Funktionen

Eingabe der Abschnitte: (Feld_Nummer, Abschnitts_Nummer, Abschnitts_Länge[m])

$$Ab := \begin{pmatrix} 1 & 1 & l_{a1} \\ 1 & 2 & l_{a2} \\ 2 & 1 & l_{a3} \end{pmatrix}$$

Eingabe der Querschnittsdefinitionen [kN*m^2]

$$EI := (E \cdot I \quad E \cdot I)$$

Eingabe des Bettungsmodul [kN/m^3]

$$Cs := (0 \quad 0)$$

Eingabe der Normalkraft Th.II Ordnung [kN]: N = 0 ( Th. I Ordnung )

$$N := (0 \quad 0)$$

Lagerdefinition am linken Trägerrand

$$Lli := 1$$

Lagerdefinition am rechten Trägerrand

$$Lre := 1$$

Eingabe der Abschnittslasten: Feld_Nr, Abschnitts_Nr, LF_Nr, q_li [kN/m], q_re [kN/m]

$$Ab_Last := \begin{pmatrix} 1 & 1 & 1 & g_d & g_d \\ 1 & 2 & 1 & g_d & g_d \\ 2 & 1 & 1 & g_d & g_d \\ 1 & 1 & 2 & q_d & q_d \\ 1 & 2 & 2 & q_d & q_d \\ 2 & 1 & 3 & q_d & q_d \end{pmatrix}$$

Eingabe der Abschnittslasten: Feld_Nr, Abschnitts_Nr, LF_Nr, F[kN], M[kN*m], W[m], Phi[rad]

$$Pkt_Last := (1 \quad 2 \quad 4 \quad F_d \quad 0 \quad 0 \quad 0)$$

## Schnittgrößenermittlung

## Berechnung der Auflagerkräfte

$$AuflKr := DLT_A_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1)$$

AuflKr =

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 22.28 | 83.827 |
| 1 | 5 | 97.667 | 208.416 |
| 2 | 11 | 23.814 | 59.497 |

## Berechnung der Momente

$$Mom := DLT_M_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1.5)$$

Mom =

| | 0 | 1 | 2 |
|---|---|---|---|
| | m | kN · m | kN · m |
| 0 | 0 | 0 | 0 |
| 1 | 2.5 | 18.201 | 142.38 |
| 2 | 2.5 | 18.201 | 142.38 |
| 3 | 4.35 | -16.616 | 50.23 |
| 4 | 4.35 | -16.616 | 50.23 |
| 5 | 6.2 | -144.226 | -63.78 |
| 6 | 6.2 | -144.226 | -63.78 |
| 7 | 7.9 | -34.608 | 23.826 |
| 8 | 7.9 | -34.608 | 23.826 |
| 9 | 9.6 | 11.608 | 78.019 |
| 10 | 9.6 | 11.608 | 78.019 |
| 11 | 11.3 | 23.144 | 70.077 |
| 12 | 11.3 | 23.144 | 70.077 |
| 13 | 11 | -0 | -0 |

Zur Bemessing maßgebende Momente in kN/m:

Feld 1 (positiv) $M_{F1} := 142.38$

Zwischenauflager (negativ) $M_{St} := 144.226$

Feld 2 (positiv) $M_{F2} := 78.019$

## Berechnung der Querkräfte:

$$Querkr := DLT_Q_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1.5)$$

Querkr =

| | 0 | 1 | 2 |
|---|---|---|---|
| | m | kN | kN |
| 0 | 0 | 22.28 | 83.827 |
| 1 | 2.5 | -7.72 | 30.077 |
| 2 | 2.5 | -36.744 | -0.898 |
| 3 | 4.35 | -74.331 | -25.287 |
| 4 | 4.35 | -74.331 | -25.287 |
| 5 | 6.2 | -114.106 | -47.487 |
| 6 | 6.2 | 50.179 | 94.31 |
| 7 | 7.9 | 29.779 | 57.76 |
| 8 | 7.9 | 29.779 | 57.76 |
| 9 | 9.6 | 9.379 | 21.21 |
| 10 | 9.6 | 9.379 | 21.21 |
| 11 | 11.3 | -22.947 | -3.414 |
| 12 | 11.3 | -22.947 | -3.414 |
| 13 | 11 | -59.497 | -23.814 |

Zur Bemessing maßgebende Querkraft in kN:

1.Auflager $Q_{St1} := 83.827$

Feld 1 $Q_{F1} := 36.774$

2.Auflager links $Q_{St2li} := 114.106$

2.Auflager rechts $Q_{St2re} := 94.31$

3.Auflager $Q_{St3} := 59.497$

## Berechnung der Durchbiegung

Durchbieg := DLT_W_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1)

Da die Durchbiegung mit charakteristischen Lasten, und nicht mit Designlasten, berechnet werden, muss, wird die hier ermittelte Durchbiegung jeweils durch den Wert 1,4 geteilt, ein Durchschnittswert der beiden Teilsicherheitsbeiwerte der Einwirkungen.

Durchbieg =

| | m | m | m |
|---|---|---|---|
| | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 |
| 1 | 1.25 | 0 | 0.004 |
| 2 | 1.25 | 0 | 0.004 |
| 3 | 2.5 | 0 | 0.006 |
| 4 | 2.5 | 0 | 0.006 |
| 5 | 3.733 | -0 | 0.005 |
| 6 | 3.733 | -0 | 0.005 |
| 7 | 4.967 | -0 | 0.003 |
| 8 | 4.967 | -0 | 0.003 |
| 9 | 6.2 | 0 | 0 |
| 10 | 6.2 | 0 | 0 |
| 11 | 7.333 | -0.001 | 0.002 |
| 12 | 7.333 | -0.001 | 0.002 |
| 13 | 8.467 | -0.001 | 0.004 |
| 14 | 8.467 | -0.001 | 0.004 |
| 15 | 9.6 | 0 | 0.005 |
| 16 | 9.6 | 0 | 0.005 |
| 17 | 10.733 | 0 | 0.004 |
| 18 | 10.733 | 0 | 0.004 |
| 19 | 11.867 | 0 | 0.003 |
| 20 | 11.867 | 0 | 0.003 |
| 21 | 11 | 0 | 0 |

Begrenzung der Verformung

allg.: $f \leq \frac{l}{250}$

$f_1 := \frac{0.006}{1.4}$

$f_1 = 0.004 \qquad < \qquad \frac{L_1}{250} = 0.025$

$f_2 := \frac{0.005}{1.4}$

$f_2 = 0.004 \qquad < \qquad \frac{L_2}{250} = 0.027$

## Eingabewerte für die Stahlbau Funktionen

Sigma_Nachweis_My(Nd, Myd, Vzd, Profil, Stahlguete, Begrenzung)

Der Nachweis wird an folgenden zwei Stellen geführt: im Feld 1, an der Stelle des höchsten Feldmomentes und am Zwischenauflager.

Nachweis im Feld:

Sigma_Nachweis_My$\left[0, M_{F1}, Q_{F1}, (2 \;\; 450), 4, 1\right] = 0.254$ < 1

Nachweis am Zwischenlager:

Sigma_Nachweis_My$\left[0, M_{St}, Q_{St2li}, (2 \;\; 450), 4, 1\right] = 0.257$ < 1

**Eingabewerte für die BDK Funktionen**

Anzusetzende Vorverformungen werden hier nicht berücksichtigt!

Nachweis für Feld 1: $\beta := 1.0$

- Ermittlung des Momentenbeiwertes für das Biegeknicken:

$\psi := 0 < 0.77$ : $\beta_{mBK} := 1$

- Ermittlung des Momentenbeiwertes für das Biegedrillknicken

$\beta_{M\psi} := 1.8 - 0.7 \cdot \psi \qquad M_Q := \frac{|M_{St}|}{2} + |M_{F1}| \quad \Delta M := |M_{St}| + |M_{F1}| \qquad \beta_{MQ} := 1.3$

$$\beta_{mBDK} := \beta_{M\psi} + \left(\frac{M_Q}{\Delta M}\right) \cdot \left(\beta_{MQ} - \beta_{M\psi}\right) \qquad \beta_{mBDK} = 1.426$$

- $zp_z := 0.225$ wenn die Querkraft am Zuggurt angreift (im Bereich der Auflager)
- $zp_z := -0.225$ wenn die Querkraft am Druckgurt angreift (im Feldbereich)
- Momentenbeiwert für die Gabellagerung $Xi := 1.12$

Nachweis im Bereich des Feldes 1:

BK BDK

$\text{biegungEinachsig}\left[\begin{pmatrix} 0 & M_{F1} \end{pmatrix}, \begin{pmatrix} 2 & 450 \end{pmatrix}, 4, L_1, \begin{pmatrix} \beta & \beta \end{pmatrix}, \beta_{mBK}, \beta_{mBDK}, zp_D, Xi\right] = \begin{pmatrix} 0.254 & 1 \end{pmatrix}$

≤1 ≤1

Nachweis im Bereich des Zwischenlagers:

$\text{biegungEinachsig}\left[\begin{pmatrix} 0 & M_{St} \end{pmatrix}, \begin{pmatrix} 2 & 450 \end{pmatrix}, 4, L_1, \begin{pmatrix} \beta & \beta \end{pmatrix}, \beta_{mBK}, \beta_{mBDK}, zp_z, Xi\right] = \begin{pmatrix} 0.257 & 0.457 \end{pmatrix}$

≤1 ≤1

Nachweis für Feld 2: $\beta := 1.0$

- Ermittlung des Momentenbeiwertes für das Biegeknicken: $\psi := 0 < 0.77$ : $\beta_{mBK} := 1$

- Ermittlung des Momentenbeiwertes für das Biegedrillknicken

$\beta_{M\psi} := 1.8 - 0.7 \cdot \psi \qquad M_Q := \frac{|M_{St}|}{2} + |M_{F2}| \quad \Delta M := |M_{St}| + |M_{F2}| \qquad \beta_{MQ} := 1.3$

$$\beta_{mBDK} := \beta_{M\psi} + \left(\frac{M_Q}{\Delta M}\right) \cdot \left(\beta_{MQ} - \beta_{M\psi}\right) \qquad \beta_{mBDK} = 1.462$$

- $zp_z := 0.225$ wenn die Querkraft am Zuggurt angreift (im Bereich der Auflager)
- $zp_z := -0.225$ wenn die Querkraft am Druckgurt angreift (im Feldbereich)
- Momentenbeiwert für die Gabellagerung $Xi := 1.12$

Nachweis im Bereich des Feldes 2:

$$\text{biegungEinachsig}\left[\begin{pmatrix}0 & M_{F2}\end{pmatrix},\begin{pmatrix}2 & 450\end{pmatrix},4,L_1,\begin{pmatrix}\beta & \beta\end{pmatrix},\beta_{mBK},\beta_{mBDK},zp_D,Xi\right] = \begin{pmatrix}0.139 & 0.548\end{pmatrix}$$

| BK | BDK |
| --- | --- |
| ≤1 | ≤1 |

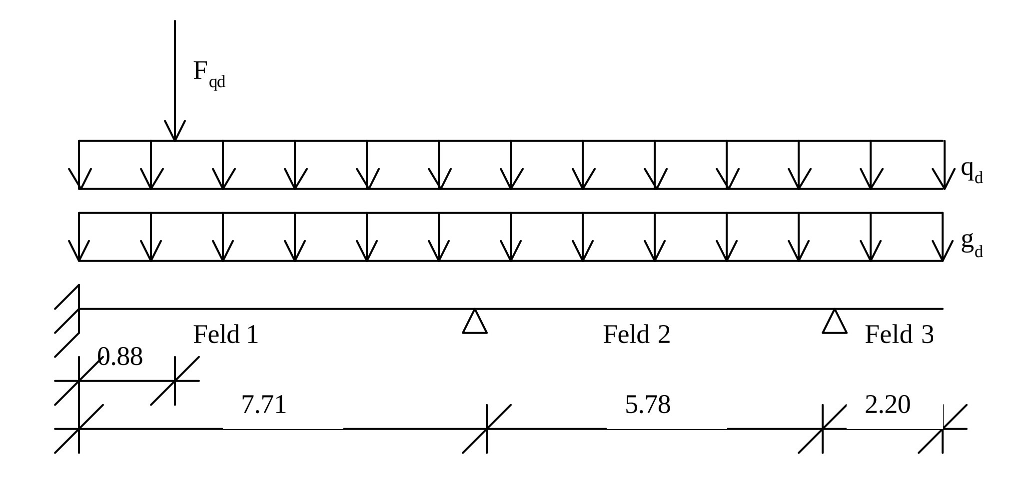
Nachweis im Bereich des Zwischenlagers:

$$\text{biegungEinachsig}\left[\begin{pmatrix}0 & M_{St}\end{pmatrix},\begin{pmatrix}2 & 450\end{pmatrix},4,L_1,\begin{pmatrix}\beta & \beta\end{pmatrix},\beta_{mBK},\beta_{mBDK},zp_z,Xi\right] = \begin{pmatrix}0.257 & 0.457\end{pmatrix}$$

| | |
| --- | --- |
| ≤1 | ≤1 |

## 5.5 Durchlaufträger in Brettschichtholz (BSH) nach DIN 1052

### Systemskizze




### Systemwerte

Feldlängen [m]

$L_1 := 7.71$

$L_2 := 5.78$

$L_3 := 2.20$

Abschnittslängen [m]

$l_{a1} := 0.88$

$l_{a2} := 6.83$

$l_{a3} := 5.78$

$l_{a4} := 2.20$

Querschnittswerte: Breite in m $b := 0.16$ Höhe in m $h := 0.50$

Eigenlast in kN/m $g_E := b \cdot h \cdot 5$ $g_E = 0.4$

Trägheitsmodul in m^4 $I := 1.042 \cdot 10^{-3}$

Material: - Brettschichtholz aus Nadelholz, Festgkeitsklasse BS 11, Sortierklasse der Lamellen S10

Elastizitätsmodul in kN/m² $E_{II} := 11000000$ $E_{II} \cdot I = 11462$ kN/m²

**Belastungszusammenstellung**

Ständige Lasten: -Eigengewicht in kN/m $g_E = 0.4$

-sonstige ständigeLasten in kN/m $g_s := 3$

Summe der ständigen Lasten in kN/m $g := g_E + g_s$ $g = 3.4$

Verkehrslast: Streckenlast in kN/m $q := 6$ Einzellast in kN $F_q := 20$

**Eingabewerte für die DLT-Funktionen**

Zeilenweise Eingabe der Abschnitte
(Feld_Nummer, Abschnitts_Nummer, Abschnitts_Länge[m]

$$Ab := \begin{pmatrix} 1 & 1 & l_{a1} \\ 1 & 2 & l_{a2} \\ 2 & 1 & l_{a3} \\ 3 & 1 & l_{a4} \end{pmatrix}$$

Feldweise Eingabe der Querschnittsdefinitionen [kN*m^2]

$$EI := \begin{pmatrix} E_{II} \cdot I & E_{II} \cdot I & E_{II} \cdot I \end{pmatrix}$$

Feldweise Eingabe des Bettungsmodul [kN/m^3]

$$Cs := \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

Feldweise Eingabe der Normalkraft Th.II Ordnung [kN]
N = 0 ( Th. I Ordnung )

$$N := \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

Lagerdefinition am linken Trägerrand

$Lli := 2$

Lagerdefinition am rechten Trägerrand

$Lre := 3$

Zeilenweise Eingabe der Abschnittslasten
Feld_Nummer, Abschnitts_Nummer, Lastfall_Nummer, q_li [kN/m], q_re [kN/m]

$$Ab_Last := \begin{pmatrix} 1 & 1 & 1 & g & g \\ 1 & 2 & 1 & g & g \\ 2 & 1 & 1 & g & g \\ 3 & 1 & 1 & g & g \\ 1 & 1 & 2 & q & q \\ 1 & 2 & 2 & q & q \\ 2 & 1 & 3 & q & q \\ 3 & 1 & 4 & q & q \end{pmatrix}$$

Zeilenweise Eingabe der Abschnittslasten
Feld_Nummer, Abschnitts_Nummer, Lastfall_Nummer, F[kN], M[kN*m], W[m], Phi[rad]

$$Pkt_Last := \begin{pmatrix} 1 & 2 & 5 & F_q & 0 & 0 & 0 \end{pmatrix}$$

**Schnittgrößenermittlung**

**Berechnung der Auflagerkräfte**

AuflKr := DLT_A_MinMax(Ab , EI , Cs , N , Lli , Lre , Ab_Last , Pkt_Last , 0.1)
Die Funktion liefert die minimalen und maximalen Auflagerkräfte in einer n*m Matrix.

AuflKr =

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 11.327 | 59.942 |
| 1 | 6 | 19.336 | 68.64 |
| 2 | 11 | 13.674 | 47.913 |
| 3 | 13 | 0 | 0 |

## Berechnung der Momente

$Mom := DLT_M_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1)$

Die Funktion liefert die minimalen und maximalen Biegemomente über den Trägerverlauf in einer n*m Matrix.

Mom =

| | 0 (m) | 1 (kN·m) | 2 (kN·m) |
|---|---|---|---|
| 0 | 0 | -71.754 | -12.269 |
| 1 | 0.88 | -25.505 | -0.757 |
| 2 | 0.88 | -25.505 | -0.757 |
| 3 | 2.018 | 1.934 | 9.05 |
| 4 | 2.018 | 1.934 | 9.05 |
| 5 | 3.157 | 6.548 | 25.096 |
| 6 | 3.157 | 6.548 | 25.096 |
| 7 | 4.295 | 5.022 | 30.696 |
| 8 | 4.295 | 5.022 | 30.696 |
| 9 | 5.433 | -0.909 | 24.115 |
| 10 | 5.433 | -0.909 | 24.115 |
| 11 | 6.572 | -11.512 | 5.618 |
| 12 | 6.572 | -11.512 | 5.618 |
| 13 | 7.71 | -41.744 | -9.835 |
| 14 | 7.71 | -41.744 | -9.835 |
| 15 | 8.866 | -15.934 | 2.686 |
| 16 | 8.866 | -15.934 | 2.686 |
| 17 | 10.022 | -10.822 | 18.8 |
| 18 | 10.022 | -10.822 | 18.8 |
| 19 | 11.178 | -10.254 | 22.352 |
| 20 | 11.178 | -10.254 | 22.352 |
| 21 | 12.334 | -14.229 | 13.343 |
| 22 | 12.334 | -14.229 | 13.343 |
| 23 | 13.49 | -22.748 | -8.228 |
| 24 | 13.49 | -22.748 | -8.228 |
| 25 | 14.59 | -5.687 | -2.057 |
| 26 | 14.59 | -5.687 | -2.057 |
| 27 | 13 | $-5.329\cdot10^{-14}$ | $-1.754\cdot10^{-14}$ |

Zur Bemessing maßgebende Momente in kN/m:

1.Auflager $M_{St1} := 71.754$

Feld 1 $M_{F1} := 30.969$

2.Auflager $M_{St2} := 41.744$

Feld 2 $M_{F2} := 22.352$

3.Auflager $M_{St3} := 22.748$

Kragarm $M_{F3} := 22.748$

## Berechnung der Querkräfte

Querkr := DLT_Q_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1)

Die Funktion liefert die minimalen und maximalen Querkräfte über den Trägerverlauf in einer n*m Matrix.

Querkr =

| | m | kN | kN |
|---|---|---|---|
| | 0 | 1 | 2 |
| 0 | 0 | 11.327 | 59.942 |
| 1 | 0.88 | 8.335 | 51.67 |
| 2 | 0.88 | 7.786 | 32.219 |
| 3 | 2.018 | 3.916 | 21.519 |
| 4 | 2.018 | 3.916 | 21.519 |
| 5 | 3.157 | 0.046 | 10.818 |
| 6 | 3.157 | 0.046 | 10.818 |
| 7 | 4.295 | -3.825 | 0.118 |
| 8 | 4.295 | -3.825 | 0.118 |
| 9 | 5.433 | -14.274 | -4.003 |
| 10 | 5.433 | -14.274 | -4.003 |
| 11 | 6.572 | -24.975 | -7.873 |
| 12 | 6.572 | -24.975 | -7.873 |
| 13 | 7.71 | -35.675 | -11.744 |
| 14 | 7.71 | 7.592 | 32.965 |
| 15 | 8.866 | 3.662 | 22.098 |
| 16 | 8.866 | 3.662 | 22.098 |
| 17 | 10.022 | -0.269 | 11.232 |
| 18 | 10.022 | -0.269 | 11.232 |
| 19 | 11.178 | -5.5 | 1.666 |
| 20 | 11.178 | -5.5 | 1.666 |
| 21 | 12.334 | -16.367 | -2.264 |
| 22 | 12.334 | -16.367 | -2.264 |
| 23 | 13.49 | -27.233 | -6.194 |
| 24 | 13.49 | 7.48 | 20.68 |
| 25 | 14.59 | 3.74 | 10.34 |
| 26 | 14.59 | 3.74 | 10.34 |
| 27 | 13 | $-1.066 \cdot 10^{-14}$ | $-1.721 \cdot 10^{-15}$ |

Zur Bemessing maßgebende Querkraft in kN:

1.Auflager $Q_{St1} := 59.942$

2.Auflager links $Q_{St2li} := 35.675$

2.Auflager rechts $Q_{St2re} := 32.965$

3.Auflager links $Q_{St3li} := 27.233$

3.Auflager rechts $Q_{St3re} := 20.68$

**Berechnung der Durchbiegung**

Durchbieg := DLT_W_MinMax(Ab, EI, Cs, N, Lli, Lre, Ab_Last, Pkt_Last, 1)

Die Funktion liefert die minimale und maximale Durchbiegung über den Trägerverlauf in einer n*m Matrix.

Durchbieg =

| | m | m | m |
|---|---|---|---|
| | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.88 | 0 | 0.002 |
| 2 | 0.88 | 0 | 0.002 |
| 3 | 2.018 | 0.001 | 0.007 |
| 4 | 2.018 | 0.001 | 0.007 |
| 5 | 3.157 | 0.001 | 0.011 |
| 6 | 3.157 | 0.001 | 0.011 |
| 7 | 4.295 | 0.001 | 0.012 |
| 8 | 4.295 | 0.001 | 0.012 |
| 9 | 5.433 | 0 | 0.01 |
| 10 | 5.433 | 0 | 0.01 |
| 11 | 6.572 | -0.001 | 0.005 |
| 12 | 6.572 | -0.001 | 0.005 |
| 13 | 7.71 | 0 | 0 |
| 14 | 7.71 | 0 | 0 |
| 15 | 8.866 | -0.003 | 0.003 |
| 16 | 8.866 | -0.003 | 0.003 |
| 17 | 10.022 | -0.004 | 0.005 |
| 18 | 10.022 | -0.004 | 0.005 |
| 19 | 11.178 | -0.004 | 0.006 |
| 20 | 11.178 | -0.004 | 0.006 |
| 21 | 12.334 | -0.003 | 0.004 |
| 22 | 12.334 | -0.003 | 0.004 |
| 23 | 13.49 | 0 | 0 |
| 24 | 13.49 | 0 | 0 |
| 25 | 14.59 | -0.003 | 0.005 |
| 26 | 14.59 | -0.003 | 0.005 |
| 27 | 13 | -0.006 | 0.01 |

Begrenzung der Durchbiegung nach DIN 1052-1:

$$f_1 := 0.012 \quad < \frac{L_1}{300} = 0.026$$

$$f_2 := 0.006 \quad < \frac{L_2}{300} = 0.019$$

$$f_3 := 0.01 \quad < \frac{L_3}{150} = 0.015$$

**Spannungsnachweise**

Holzbau-Funktion: HolzDruckMNachweis

HolzDruckMNachweis(My[kNm],N[Kn],s[m],sKy[m],sKz[m],h[m],b[m],HolzKl))

Diese Funktion wurde gewählt, da es keine Funktion gibt, die nur Biegung berücksichtigt. Die Funktion HolzDruckMNachweis berücksichtigt auch die Kippsicherheit.

Feld 1: $\text{HolzDruckMNachweis}\left(M_{St1}, 0, L_1, 0.7 \cdot L_1, 0.7 \cdot L_1, h, b, 9\right) = 0.978$ < 1

Feld 2: $\text{HolzDruckMNachweis}\left(M_{St2}, 0, L_2, 1.0 \cdot L_2, 1.0 \cdot L_2, h, b, 9\right) = 0.569$ < 1

Feld 3: $\text{HolzDruckMNachweis}\left(M_{St3}, 0, L_3, 2 \cdot L_3, 2 \cdot L_3, h, b, 9\right) = 0.31$ < 1

**Schubnachweis**

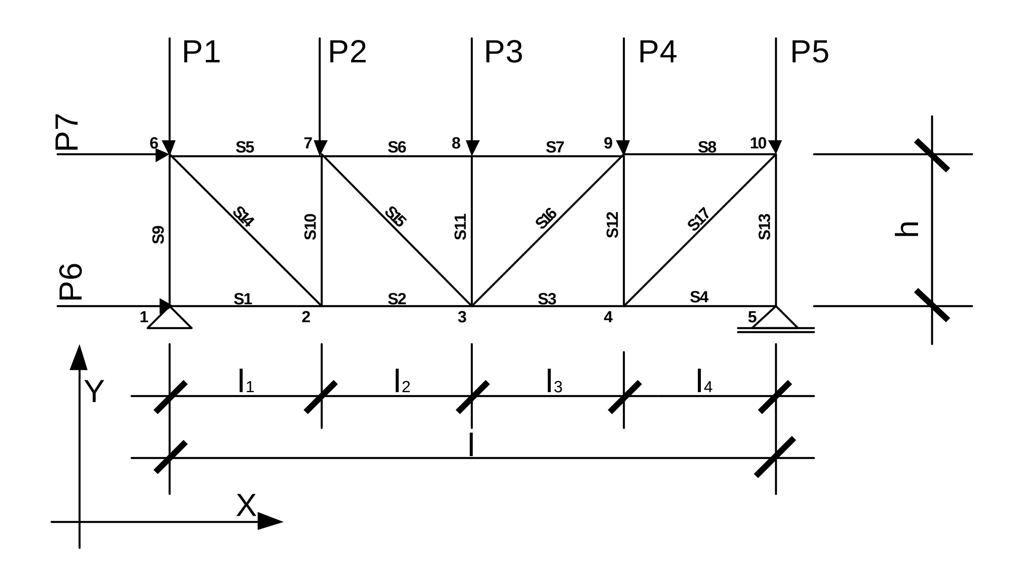
- maximaler Schub aus Querkraft in MN/m²: $\tau_Q := 1.5 \cdot \frac{Q_{St1} \cdot 10^{-3}}{b \cdot h}$ $\tau_Q = 1.124$

- zulässige Schubspannung in MN/m²: $zul\tau_Q := 1.2$

Nachweis: $\frac{\tau_Q}{zul\tau_Q} = 0.937$ < 1

## 5.6 Ständerfachwerk mit fallenden Streben in Holz nach DIN 1052

### Systemskizze




$l_1 = l_2 = l_3 = l_4 = h = 2$ m

### Eingabe der Systemknoten

Es können maximal 25 Knoten eingegeben werden.
Die Knoteneingabe erfolgt spaltenweise (Nummerierung, Koordinaten und Lagerbedingungen siehe Systemskizze.):

1.Zeile: Knotennummer (1, 2, ...)
2.Zeile: x-Koordinate in m
3.Zeile: y-Koordinate in m
4.Zeile: Auflager in x-Richtung: 1:ja - 0:nein
5.Zeile: Auflager in y-Richtung: 1:ja - 0:nein

$$\text{Knoten} := \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 0 & 2 & 4 & 6 & 8 & 0 & 2 & 4 & 6 & 8 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

### Eingabe der Querschnitts- und Materialliste

Querschnitt (verwendet wird ein 228 cm² großer Querschnitt (12*24) aus NH S10.):

1.Zeile: Materialnummer (1, 2, ...)
2.Zeile: Fläche (in qm)
3.Zeile: e-Modul (in KN/qm)
4.Zeile: eine optionale Materialbenennung

$$\text{Querschnitt} := \begin{pmatrix} 1 \\ 0.0228 \\ 1000000 \end{pmatrix}$$

### Eingabe der System-Stäbe

(maximal 25)

Stäbe (alle Stäbe weisen den selben Querschnitt auf, Nummerierung siehe Systemskizze):

1.Zeile: Stabnummer (1, 2, ...)
2.Zeile: Anfangsknoten
3.Zeile: Endknoten
4.Zeile: Materialnummer

$$\text{Stäbe} := \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 \\ 1 & 2 & 3 & 4 & 6 & 7 & 8 & 9 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 9 & 10 \\ 2 & 3 & 4 & 5 & 7 & 8 & 9 & 10 & 6 & 7 & 8 & 9 & 10 & 2 & 3 & 3 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

## Eingabe der Knotenlasten

Die Eingabe der Kräfte erfolgt Lastfallweise.

Lastfall 1: Belastung aus ständigen Lasten - Vertikale Knotenlasten in den Knoten 6, 7, 8, 9 und 10

Knotenlasten:
1.Zeile: Knotennummer (beliebige Reihenfolge)
2.Zeile: Kraft in x-Richtung in kN
3.Zeile: Kraft in y-Richtung in kN

$$\text{Lastfall1} := \begin{pmatrix} 6 & 7 & 8 & 9 & 10 \\ 0 & 0 & 0 & 0 & 0 \\ 10 & 20 & 20 & 20 & 10 \end{pmatrix}$$

Lastfall 2 : Belastung aus Verkehrslasten - Vertikale Knotenlasten in den Knoten 5, 7, 8
Horizontale Knotenlasten in den Knoten 3,5,7

$$\text{Lastfall2} := \begin{pmatrix} 3 & 5 & 7 & 8 \\ 10 & 20 & 10 & 0 \\ 0 & 10 & 20 & 10 \end{pmatrix}$$

Lastfall 3: Belastung aus Verkehrslasten - Vertikale Knotenlasten in den Knoten 1, 2, 4
Horizontale Knotenlasten in den Knoten 1, 4

$$\text{Lastfall3} := \begin{pmatrix} 2 & 4 & 1 \\ 0 & 10 & 20 \\ 10 & 20 & 30 \end{pmatrix}$$

## Ermittlung der Stabkräfte

Fachwerk-Funktion Stabwerk_Stabkräfte: Als Ergebnis wird eine n*m Matrix ausgegeben, welche die Stabkräfte für den jeweiligen Lastfall enthält.

StabkrLF1 := Stabwerk_Stabkräfte(Knoten, Querschnitt, Stäbe, Lastfall1)
StabkrLF2 := Stabwerk_Stabkräfte(Knoten, Querschnitt, Stäbe, Lastfall2)
StabkrLF3 := Stabwerk_Stabkräfte(Knoten, Querschnitt, Stäbe, Lastfall3)

Zur Lösung: 1.Spalte - Stabnummer
2.Spalte - Stabkraft in kN (Druck - positiv, Zug - negativ)

$StabkrLF1^T =$

| | 1 |
|---|---|
| 1 | 0 |
| 2 | -30 |
| 3 | -30 |
| 4 | 0 |
| 5 | 30 |
| 6 | 40 |
| 7 | 40 |
| 8 | 30 |
| 9 | 40 |
| 10 | 30 |
| 11 | 20 |
| 12 | 30 |
| 13 | 40 |
| 14 | -42.426 |
| 15 | -14.142 |
| 16 | -14.142 |
| 17 | -42.426 |

$StabkrLF2^T =$

| | 1 |
|---|---|
| 1 | 40 |
| 2 | 17.5 |
| 3 | 12.5 |
| 4 | 20 |
| 5 | 22.5 |
| 6 | 15 |
| 7 | 15 |
| 8 | 7.5 |
| 9 | 22.5 |
| 10 | 22.5 |
| 11 | 10 |
| 12 | 7.5 |
| 13 | 7.5 |
| 14 | -31.82 |
| 15 | -3.536 |
| 16 | -10.607 |
| 17 | -10.607 |

$StabkrLF3^T =$

| | 1 |
|---|---|
| 1 | 10 |
| 2 | -2.5 |
| 3 | -7.5 |
| 4 | 0 |
| 5 | 12.5 |
| 6 | 15 |
| 7 | 15 |
| 8 | 17.5 |
| 9 | 12.5 |
| 10 | 2.5 |
| 11 | -0 |
| 12 | -2.5 |
| 13 | 17.5 |
| 14 | -17.678 |
| 15 | -3.536 |
| 16 | 3.536 |
| 17 | -24.749 |

**Berechnung von Lastfallüberlagerungen**

Vorgehensweise:
Als erstes wird die Ergebnismatrix aus Lastfall 1 (ständige Lasten) mit der Ergebnismatrix aus Lastfall 2 (Verkehrslasten) überlagert. Das Ergebnis dieser Überlagerung wird wiederum mit der Ergebnismatrix aus Lastfall 3 (Verkehrslasten) überlagert. Somit erhält man die Maximal- und Minimalwerte der Stabkräfte aller Lastfallkombinationen um damit die erforderlichen Nachweise zu führen.

Fachwerk-Funktion Ueberlag_N_LFG_LFPn:
Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Stabkräfte in einer n*m Matrix zeilenweise die Überlagerung der MIN/MAX Werte der Stabkräfte, Max_N, Min_N.

$Ueberlag_N_LFG_LFPn(StabkrLF1, StabkrLF2)^T =$

| | 1 | 2 |
|---|---|---|
| 1 | 40 | 0 |
| 2 | -12.5 | -30 |
| 3 | -17.5 | -30 |
| 4 | 20 | 0 |
| 5 | 52.5 | 30 |
| 6 | 55 | 40 |
| 7 | 55 | 40 |
| 8 | 37.5 | 30 |
| 9 | 62.5 | 40 |
| 10 | 52.5 | 30 |
| 11 | 30 | 20 |
| 12 | 37.5 | 30 |
| 13 | 47.5 | 40 |
| 14 | -42.43 | -74.25 |
| 15 | -14.14 | -17.68 |
| 16 | -14.14 | -24.75 |
| 17 | -42.43 | -53.03 |

$Ergebnis1 := Ueberlag_N_LFG_LFPn(StabkrLF1, StabkrLF2)$

Fachwerk-Funktion Ueberlag_N_LFG+LFPn_LFPn:
Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Stabkräfte und der Lösung aus Ueberlag_N_LFG_LFPn in einer n*m Matrix zeilenweise die MIN/MAX Werte der Stabkräfte, Max_N, Min_N.

$Ueberlag_N_LFG+LFPn_LFPn(Ergebnis1, StabkrLF3)^T =$

| | 1 | 2 |
|---|---|---|
| 1 | 50 | 0 |
| 2 | -12.5 | -32.5 |
| 3 | -17.5 | -37.5 |
| 4 | 20 | 0 |
| 5 | 65 | 30 |
| 6 | 70 | 40 |
| 7 | 70 | 40 |
| 8 | 55 | 30 |
| 9 | 75 | 40 |
| 10 | 55 | 30 |
| 11 | 30 | 20 |
| 12 | 37.5 | 27.5 |
| 13 | 65 | 40 |
| 14 | -42.43 | -91.92 |
| 15 | -14.14 | -21.21 |
| 16 | -10.61 | -24.75 |
| 17 | -42.43 | -77.78 |

Es wird mit Funktionen der Kategorie *Bem_Funktionen* für Zug- und Druckstäbe aus Holz nachgewiesen, dass die zulässigen Normalkräfte in der Matrix oben nicht überschritten werden.

**Eingabewerte für die Bem_Funktionen**

Breite des Querschnittes in cm b := 12 Höhe des Querschnittes in cm h := 24

Holzart NH S10 Holzart := 2

Knicklängen der Stäbe 1 bis 13 in cm: sky_kurz := 200 skz_kurz := 200

Maximale Druckkraft der Stäbe 1 bis 13 in kN: Druck_kurz := 75

Maximale Zugkraft der Stäbe 1 bis 13 in kN: Zug_kurz := 37.5

Knicklängen der Stäbe 14 bis 17 in cm: sky_lang := 283 skz_lang := 283

Maximale Druckkraft der Stäbe 14 bis 17 in kN: Druck_lang := 0

Maximale Zugkraft der Stäbe 14 bis 17 in kN: Zug_lang := 91.92

Funktion:
HolzDruck_Nachweis(N[kN],sky[cm],skz[cm],Holzart[1-12],b[cm],h[cm])

Nachweis_Druck_kurzeStäbe := HolzDruck_Nachweis(Druck_kurz, sky_kurz, skz_kurz, Holzart, b, h)

Nachweis_Druck_kurzeStäbe = 0.5: < 1

Nachweis_Druck_langeStäbe := HolzDruck_Nachweis(Druck_lang, sky_lang, skz_lang, Holzart, b, h)

Nachweis_Druck_langeStäbe = ( < 1

Funktion: HolzZugNachweisV1(N[kN],b[cm],h[cm],Holzart[1-12])

Nachweis_Zug_kurzeStäbe := HolzZugNachweisV1(Zug_kurz, b, h, Holzart)

Nachweis_Zug_kurzeStäbe = 0.18( < 1

Nachweis_Zug_langeStäbe := HolzZugNachweisV1(Zug_lang, b, h, Holzart)

Nachweis_Zug_langeStäbe = 0.456 < 1

**Ermittlung der Knotenverschiebungen**

Fachwerk-Funktion Stabwerk_Verschiebungen: Als Ergebnis werden in einer n*m Matrix die Verschiebungen in x- und y-Richtung für den jeweiligen Lastfall ausgegeben.

KnVerschLF1 := Stabwerk_Verschiebungen(Knoten, Querschnitt, Stäbe, Lastfall1)

KnVerschLF2 := Stabwerk_Verschiebungen(Knoten, Querschnitt, Stäbe, Lastfall2)

KnVerschLF3 := Stabwerk_Verschiebungen(Knoten, Querschnitt, Stäbe, Lastfall3)

Zur Lösung:

1.Zeile: Knotennummer
2.Zeile: Verschiebung in x-Richtung in mm
3.Zeile: Verschiebung in y-Richtung in mm

KnVerschLF1 =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | -2.63 | -5.26 | -5.26 | -8.77 | -6.14 | -2.63 | 0.88 | 3.51 |
| 2 | 0 | 19.72 | 28.35 | 19.72 | 0 | 3.51 | 22.36 | 30.1 | 22.36 | 3.51 |

KnVerschLF2 =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 3.51 | 5.04 | 6.14 | 7.89 | 3.25 | 5.23 | 6.54 | 7.86 | 8.52 |
| 2 | 0 | 7.81 | 10.23 | 4.89 | 0 | 1.97 | 9.79 | 11.1 | 5.55 | 0.66 |

KnVerschLF3 =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.88 | 0.66 | 0 | 0 | -1.97 | -0.88 | 0.44 | 1.75 | 3.29 |
| 2 | 0 | 7.05 | 9.42 | 9.17 | 0 | 1.1 | 7.27 | 9.42 | 8.95 | 1.54 |

**Berechnung von Lastfallüberlagerungen**

Vorgehensweise:
Als erstes wird die Ergebnissmatrix aus Lastfall 1 (ständige Lasten) mit der Ergebnissmatrix aus Lastfall 2 (Verkehrslasten) überlagert. Das Ergebniss dieser Überlagerung wird wiederum mit der Ergebnissmatrix aus Lastfall 3 (Verkehrslasten) überlagert. Somit erhält man die Maximal- und Minimalwerte der Knotenverschiebungen aller Lastfallkombinationen.

Fachwerk-Funktion Ueberlag_w_LFG_LFPn:
Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Verschiebung in einer n*m Matrix zeilenweise die Überlagerung der MIN/MAX Werte der Knotenverschiebungen, Max_x, Min_x, Max_y, Min_y.

$\text{Ueberlag_w_LFG_LFPn}(\text{KnVerschLF1}, \text{KnVerschLF2})^T =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 3.51 | 0 | 27.54 | 19.72 |
| 3 | 2.41 | -2.63 | 38.57 | 28.35 |
| 4 | 0.88 | -5.26 | 24.62 | 19.72 |
| 5 | 2.63 | -5.26 | 0 | 0 |
| 6 | -5.52 | -8.77 | 5.48 | 3.51 |
| 7 | -0.91 | -6.14 | 32.14 | 22.36 |
| 8 | 3.91 | -2.63 | 41.2 | 30.1 |
| 9 | 8.73 | 0.88 | 27.91 | 22.36 |
| 10 | 12.02 | 3.51 | 4.17 | 3.51 |

$\text{Ergebnis2} := \text{Ueberlag_w_LFG_LFPn}(\text{KnVerschLF1}, \text{KnVerschLF2})$

Fachwerk-Funktion Ueberlag_w_LFG+LFPn_LFPn:

Die Funktion liefert aus den Ergebnismatrizen der Funktion Stabwerk_Verschiebung und der Lösung aus Ueberlag_w_LFG_LFPn in einer n*m Matrix zeilenweise die MIN/MAX Werte der Knotenverschiebungen, Max_x, Min_x, Max_y, Min_y.

$$\text{Ueberlag_w_LFG+LFPn_LFPn}(\text{Ergebnis2}, \text{KnVerschLF3})^T =$$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 4.39 | 0 | 34.59 | 19.72 |
| 3 | 3.07 | -2.63 | 47.99 | 28.35 |
| 4 | 0.88 | -5.26 | 33.78 | 19.72 |
| 5 | 2.63 | -5.26 | 0 | 0 |
| 6 | -5.52 | -10.75 | 6.58 | 3.51 |
| 7 | -0.91 | -7.02 | 39.41 | 22.36 |
| 8 | 4.35 | -2.63 | 50.63 | 30.1 |
| 9 | 10.49 | 0.88 | 36.85 | 22.36 |
| 10 | 15.31 | 3.51 | 5.7 | 3.51 |

# Anhang A Literatur- und Internetrecherche zu Arbeitsblättern in MS-Excel für die Tragwerksplanung

## 1. Statik

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Technische Mechanik<br>Stab- und Auflagerkräfte eines Fachwerks 1<br>Auflagerkräfte eines statisch bestimmten Systems<br>Flächenschwerpunkt zusammengesetzter Flächen<br>Momente und Kräfte einer Reibungsbremse<br>Flächen- und Widerstandsmomente<br>Torsionsfedern mit runden Querschnitten<br>Biegebeanspruchung eines Balkens 3<br>Zug-, Abscherspannung und Flächenpressung<br>Biegung und Torsion von Wellen<br>Waagrechter und schräger Wurf<br>Energieerhaltungssatz: Abbremsvorgänge<br>Prinzip von d´Alembert: Massenbeschleunigung<br>Gerader zentrischer Stoß | Hans-Jürgen Holland,<br>Berechnungsbibliothek Bauwesen,<br>vieweg technic tools,<br>Technische Mechanik I,<br>EXCEL - Programme | [1]<br>[M1] | http://www.vieweg.de/ |
| | Festigkeitslehre<br>Ermittlung von Querschnittswerten I<br>Ermittlung von Querschnittswerten II<br>Ermittlung von Biegenormalspannungen<br>Ermittlung der ersten Kernfläche<br>Ermittlung von Biegenormalspannungen bei versagender Zugzone<br>Ermittlung von Biegenormalspannungen in Stahlbetonbalken<br>Ermittlung von Torsionsschubspannungen<br>Der zweiachsige Spannungszustand<br>Der dreiachsige Spannungszustand<br>Spannungsermittlung mit Dehnungsmessstreifen | Volker Slowik ,<br>Berechnungsbibliothek Bauwesen,<br>vieweg technic tools,<br>Festigkeitslehre für Bauingenieure,<br>EXCEL - Programme | [1]<br>[M3] | http://www.vieweg.de/ |
| | Mathematik<br>Quadratische Gleichungen<br>Rechnen mit komplexen Zahlen<br>Matrizenrechnung<br>Gaußscher Algorithmus zur Lösung linearer Gleichungssysteme<br>Fehlerrechnung<br>Lagrangeinterpolation<br>Newtonsches Verfahren zur näherungsweisen Bestimmung von Nullstellen einer Gleichung<br>Vereinfachtes Newtonsches Verfahren<br>Numerische Integration nach der Simpsonschen Regel<br>Punktweise Lösung einer Differentialgleichung 2.Ordnung nach dem Runge-Kutta-Verfahren<br>Digitale Fourier-Transformation zur Erstellung eines Power-Spektrums<br>Übung: Erstellung eines eigenen Arbeitsblattes (Vektorrechnung) | Gerd Küveler,<br>Berechnungsbibliothek Bauwesen<br>vieweg technic tools<br>EXCEL - Programme | [1] | |

# 2. Stahlbeton

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Stahlbeton und Spannbeton nach DIN 1045-1<br>Normtext<br>Interaktive Bemessungshilfe<br>Nachweise im Grenzzustand der Tragfähigkeit und im Grenzzustand der Gerbrauchstauglichkeit<br>Beispiele aus dem Hochbau | Schmitz / Goris<br>DIN 1045-1 digital<br>MS- Excel - Anwendungen | [2] | |
| | Massivbau - Bemessung nach Eurocode 2<br>Formel- und Kurzzeichen<br>Bemessung für Biegung mit Längskraft nach EC 2; Rechteckquerschnitt ohne Druckbewehrung<br>Bemessung für Biegung mit Längskraft nach EC 2; Rechteckquerschnitt mit Druckbewehrung<br>Bemessung für Biegung mit Längskraft nach EC 2; Plattenbalkenquerschnitt ohne Druckbewehrung<br>Bemessung für Querkraft nach EC 2; Balken mit Bügelbewehrung, Biegung mit Längskraft<br>Bemessung für Durchstanzen nach EC 2; Kreis- und Rechteckquerschnitt<br>Bemessung unbewehrter Druckglieder nach EC 2; Rechteckquerschnitt<br>Mindestbewehrung mit direkter Berechnung nach EC 2; Rechteckquerschnitt, langandauernd direkte Zwangbeanspruchung<br>Rissbreitenbeschränkung mit direkter Berechnung nach EC 2; Rechteckquerschnitt, reine Biegung<br>Nachweis der Spannungen im Zustand II nach EC 2; Rechteckquerschnitt, reine Biegung<br>Nachweis der Biegeschlankheit nach EC 2; Rechteckquerschnitt, reine Biegung<br>Nachweis der Durchbiegungen nach EC 2; Rechteckquerschnitt, ständige, gleichförmige Belastung<br>Nachweis der Durchbiegungen nach EC 2; Plattenbalkenquerschnitt, gleichförmige Belastung | Christian Findeisen, Berechnungsbibliothek Bauwesen, Vieweg –technic – tools, Massivbau, Excel-Programme | [1]<br>[M2] | http://www.vieweg.de/ |

# 3. Stahlbau

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Berechnung von Schraubenverbindungen<br>mit sämtlichen Tragsicherheitsnachweisen<br>einschließlich der Überprüfung der Mindestmaße (<br>Lochabstände ). | Bernd Humbert,<br>Fachbeitrag<br>Excel - Arbeitsblatt<br>bi – Journal 5/99 | [3]<br>[M9] | |
| | Statik im Stahlbau<br>Einfeldträger, Schnittgrößen bei Belastung mit n Einzellasten<br>Träger auf zwei Stützen, Nachweis nach DIN 18800-1: 1990<br>Kragträger, Tragsicherheitsnachweis<br>Biegedrillknicken, Nachweis nach DIN 18800 T2: 1990<br>Planmäßig mittiger Druck, Biegeknicken<br>Einachsige Biegung mit planmäßig mittigem Druck<br>Zweiachsige Biegung mit Normalkraft, Biegeknicken<br>Zweiachsige Biegung mit Normalkraft, Biegedrillknicken<br>Halsnähte, Nachweis bei geschweißten Vollwandträgern<br>Trägerstoß, Nachweis im Grundquerschnitt<br>Kopfplattenanschluß, Tragsicherheitsnachweis<br>Stoß eines Zugstabes<br>Rippenlose Krafteinleitung, Träger auf Träger<br>I-Profile, geschweißt, Abmessungen und statische Werte<br>Kastenprofile, Abmessungen und statische Werte<br>Formeln für gewalzte, doppelsymmetrische I-Profile | Erwin Piechatzek<br>Berechnungsbibliothek Bauwesen<br>Vieweg –technic - tools | [1] | |

# 4. Geotechnik

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Geotechnik<br>Erkunden – Untersuchen – Berechnen - Messen<br>Bruchmechanismus<br>Erdruck<br>Grundbau –Trainer<br>Pfähle<br>Proben<br>Setzungen<br>Wasser im Boden | Konrad Kunsche,<br>Geotechnik,<br>Excel-Arbeitsblätter,<br>Vieweg –Verlag | [4] | |

## 5. Mauerwerksbau

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Mauerwerk ( DIN 1053)<br>Rechenbeispiel nach Schneider, Klaus-Jürgen, Mauerwerksbau<br>Düsseldorf: Werner Verlag 5.Auflage S.193 | Günther Schulz,<br>Fachbeitrag<br>Excel - Arbeitsblatt<br>bi – Journal 4/99 | [5] | |
| | Mauerwerk ( DIN 1053-1)<br>Algorithmus zur Bemessung von zweiseitig gehaltenen Wänden | Günther Schulz,<br>Fachbeitrag<br>Excel - Arbeitsblatt<br>bi – Journal 6/98 | [6]<br>[M7] | |
| | Mauerwerk ( DIN 1053)<br>Excelrechenblatt zur Bemessung von Holzbalken | Günther Schulz,<br>Fachbeitrag<br>Excel – Arbeitsblatt<br>bi – Journal 1/98 | [7]<br>[M8] | |

## 6. Holzbau

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Holzbau (DIN 1052)<br>Dieses Arbeitsblatt zeigt, wie Sie mit MS – Excel Einfeldbalken aus Holz bemessen und nachweisen können. | Günter Schulz,<br>Fachbeitrag<br>Excel - Arbeitsblatt<br>bi – Journal 2/99 | [8] | |

## 7. HOAI[1]

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Dieser Beitrag zeigt die Algorithmen- und Programmstruktur der HOAI - Interpolation | Günter Schulz,<br>Fachbeitrag<br>Excel - Arbeitsblatt<br>bi – Journal 1/99 | [9]<br>[M4] | |
| | Dieser Beitrag stellt ein Excel – Arbeitsblatt zur Kostenschätzung nach DIN 276 vor. | Günter Schulz,<br>Fachbeitrag<br>Excel - Arbeitsblatt<br>bi – Journal 1/99 | [8]<br>[M5] | |
| | Beitrag über Excel – Formulare zur Kostenermittlung nach DIN 276(6.93) und DIN 276(4.81) | Günter Schulz,<br>Fachbeitrag<br>Excel - Arbeitsblatt<br>bi – Journal 2/99 | [10]<br>[M6] | |

[1] HOAI –Verordnung für Honorare der Architekten und Ingenieure. Die Bestimmungen gelten für die Berechnung der Entgelte für Leistungen der Architekten und der Ingenieure (Auftragnehmer). Die HOAI ist geltendes Preisrecht.

# Anhang B Literatur- und Internetrecherche zu Arbeitsblättern in Mathcad für die Tragwerksplanung

## 1. Statik

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Allgemeine Formeln für Querschnittswerte<br>Hauptspannungen<br>Schnitt- und Verschiebungsgrößen<br>Balken , Stabwerken<br>gekrümmte Balken , Bogen<br>Flachplatten<br>Stützen, Druckelemente<br>Schalen, Druckbehälter, Rohre<br>Torsion<br>Elastisches Verhalten von Stäben, Balken, Platten, Schalen<br>Dynamische und Temperaturspannungen | Warren C. Joung,<br>Roark`s Formulars for Stress and Stain,<br>Mathcad Elektronisches Buch | [M10] | info@mathsoft.co.uk |
| | Schnittgrößenermittlung<br>Träger auf zwei Stützen<br>Träger mit Gleichlast und Randmomenten | Thomas P. Magner,<br>Building Structural Design,<br>Mathcad Elektronisches Buch | [M11] | info@mathsoft.co.uk |
| | Formeln für Schnitt- und Verschiebungsgrößen | Werkle, Schneider<br>Bautabellen digital<br>Mathcad Arbeitsblätter | [11] | |
| | Elastische Biegung in einem Balken | Mathcad - Arbeitsblatt | [L1] | www.mathsoft.com/bending.mcd |
| | Querkraft- und Momentendiagramm eines Träger auf zwei Stützen mit Kragarm | Kristopher Hauck<br>Mathcad - Arbeitsblatt | [L2] | www.mathsoft.com/ssbeam.mcd |
| | Schnittgrößenermittlung eines Träger auf zwei Stützen | Thomas Magner<br>Mathcad - Arbeitsblatt | [L3] | www.mathsoft.com/smplbeam.mcd |
| | Platten und Schalen nach der Theorie von Timoshenko | Robert Hirschel<br>Mathcad - Arbeitsblatt | [L4] | www.mathsoft.com/plate_shell.mcd |
| | Statische Berechnungen mit Mathcad als Engineering Desktop Tool | Serkan Karadag<br>Diplomarbeit an der FH - Konstanz | [12] | |
| | Statische Berechnungen mit Mathcad | Günter Schulz,<br>Fachbeitrag<br>bi – Journal 1/99 | [13 ] | |
| | Differentialgleichungen mit Mathcad lösen | Horst Werkle,<br>Fachbeitrag<br>bi – Journal 2/99 | [14] | |
| | Statische Berechnungen mit Mathcad | Horst Werkle,<br>Fachbeitrag<br>bi – Journal 3/99 | [15] | |

# 2. Stahlbeton

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Bewehrung<br>Platten und Balken<br>Stützen<br>Flachplatten<br>Stützwänden<br>Einzelfundamenten<br>Materialeigenschaften, Entwicklung und Richtlinien | Thomas P. Manger,<br>Building Structural Design,<br>Mathcad Elektronisches Buch | [M11] | info@mathsoft.co.u k |
| | Stahlbeton und Spannbeton nach EC 2 Teil1<br>Bemessungsgrundlagen<br>Schnittgrößenermittlung<br>Bemessung<br>Bauliche Durchbiegung<br>Bemessungsbeispiel für einen vorgespannten Träger<br>Bemessungstafeln | Werkle, Schneider<br>Bautabellen digital<br>Mathcad Arbeitsblätter | [11] | |
| | Interaktionsdiagramm für<br>Rechteckbetonstützen<br>Rundbetonstützen | Paul H.Dunn<br>Mathcad - Arbeitsblatt | [L5] | www.mathsoft.com/<br>conccol.mcd<br>conccirc.mcd |
| | Zugbewehrung einer Vollplatte | Richard Foss<br>Mathcad - Arbeitsblatt | [L6] | www.mathsoft.com/<br>usd.mcd |
| | Konstruktion einer runden Betonwand | Bill Moore<br>Mathcad - Arbeitsblatt | [L7] | www.mathsoft.com/<br>conctank.mcd |
| | Abmessungen für eine bewehrte Betonstütze | Thomas P. Manger,<br>Mathcad - Arbeitsblatt | [L8] | www.mathsoft.com/<br>rectcols.mcd |
| | Entwicklung von Bewehrungsstäben | Thomas P. Manger,<br>Mathcad - Arbeitsblatt | [L9] | www.mathsoft.com/<br>devreinf.mcd |
| | Erforderliche Dicke einer Betonplatte im ungerissenen Zustand unter Punktbelastungen | Bill Moore<br>Mathcad - Arbeitsblatt | [L10] | www.mathsoft.com/<br>sog-t.mcd |
| | Durchstanznachweis mit Mathcad | Kai-Uwe Jacobs,<br>Ralf Avak, Fachbeitrag<br>bi – Journal 4/99 | [16] | |

# 3. Stahl

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Baustahlstützen ( ASD Norm) | Thomas P. Magner, Building Structural Design, Mathcad Elektronisches Buch | [M11] | info@mathsoft.co.uk |
| | Stahlverbund Deckenkonstruktionen | Thomas P. Magner, Building Structural Design, Mathcad Elektronisches Buch | [M11] | info@mathsoft.co.uk |
| | Stahlbau nach DIN 18 800 (11.90)<br>Grundlagen der Berechnungen<br>Trag-, Lage- und Gebrauchsträgfähigkeits-Nachweise nicht stabilitätsgefährdeter Bauteile<br>Verbindungen mit Schweißnähten<br>Verbindungen mit Schrauben | Werkle, Schneider<br>Bautabellen digital<br>**Mathcad Arbeitsblätter** | [11] | |
| | Minimierung einer Kostenfunktion für eine Schweißverbindung | Sandor Dominich<br>Mathcad - Arbeitsblatt | [L11] | www.mathsoft.com/weld01.mcd |
| | AISC – Spezielle Berechnungen mit Hilfe von Data Files | Thomas Magner, Specification for Structural Steel Buildings<br>Mathcad - Arbeitsblatt | [L12] | TomMagner@ Compuserve.com |

# 4. Geotechnik

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Bohrpfähle ohne Mantelreibung | Gaylord and Gaylord, Mathcad – Arbeitsblatt Structural Engineering Handbook | [ L13] | www.mathsoft.com/pole.mcd |
| | Bewegungen im Boden hervorgerufen durch eine Spinnenegge | M. Heinloo und K.Kaul Mathcad – Arbeitsblatt | [ L14] | www.mathsoft.com/spinharr.mcd |
| | Bohrpfähle ohne Mantelreibung | Gaylord and Gaylord, Mathcad – Arbeitsblatt Structural Engineering Handbook | [L13] | www.mathsoft.com/pole.mcd |
| 4.2 | Bewegungen im Boden hervorgerufen durch eine Spinnenegge | M. Heinloo und K.Kaul Mathcad – Arbeitsblatt | [L14] | www.mathsoft.com/spinharr.mcd |

# 5. Lastannahmen

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Lastannahmen (ASCE Norm)<br>Windlasten<br>Erdbebenlasten | Thomas P. Magner,<br>Building Structural Design,<br>Mathcad Elektronisches Buch | [M11] | info@mathsoft.co.uk |
| | Lastannahmen (ASCE Norm) Windlasten | Thomas Magner<br>Mathcad - Arbeitsblatt | [L15] | www.mathsoft.com/windload.mcd |
| | Lastannahmen von horizontalen Wellenbrechern | Thomas Schaap<br>Mathcad - Arbeitsblatt | [L16] | www.mathsoft.com/elasplas.mcd |

# 6.Mauerwerksbau

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Mauerwerk ( DIN 1053)<br>Mauerwerk nach DIN 1053-1<br>Bewehrtes Mauerwerk nach DIN 1053-2<br>Bemessung von Flachziegelstürzen | Werkle, Schneider<br>Bautabellen digital<br>Mathcad Arbeitsblätter | [11] | |

# 7. Holzbau

| | Themengebiete | Literaturhinweise | | Links |
|---|---|---|---|---|
| | Holzbau (DIN 1052)<br>Berechnungsgrundlagen<br>Durchbiegungsberechnung<br>Spannungsnachweise einteiliger Rechteckquerschnitte<br>Stabilitätsnachweise<br>Berechnung von BSH-Trägern<br>Berechnung zusammengesetzter Querschnitte<br>Verbindungen | Werkle, Schneider<br>Bautabellen digital<br>Mathcad Arbeitsblätter | [11] | **7.1** |

# Anhang C Verzeichnis von Windows API Funktionen

Alle Systemfunktionen die Programmierern zur Verfügung sind in verschiedenen Programmbibliotheken (*DLL's*) gespeichert. Die Gesamtheit dieser Bibliotheken wird Windows API genannt. Eine Liste der wichtigsten Programmbibliotheken ist in untenstehender Tabelle zu finden. Mit der Verwendung von API Funktionen bei der Programmierung bewegt man sich auf einer systemnahen Ebene. Man greift über API Funktionen auf Betriebssystemfunktionen oder auch auf eigene in C/C++ programmierte Funktionen zu. Ein wesentlicher Vorteil ist die Geschwindigkeit mit der auf dieser Ebene Daten verarbeitet werden können. Bei Problemen, welche die *Win 32 API* betreffen wird von vielen Seiten und auch in der Literatur auf *Dan Applemans; Visual Basic Programmer's Guide to the Win 32 API* verwiesen.

| **Dynamic-Link Library** | **Beschreibung** |
|---|---|
| Advapi32.dll | Bibliothek für erweiterte API-Dienste, die zahlreiche APIs unterstützt, darunter viele Sicherheits- und Registrierungsaufrufe |
| Comdlg32.dll | Bibliothek mit Standarddialog-APIs |
| Gdi32.dll | Bibliothek mit GDI-API (GDI = Graphics Device Interface) Enthält Zeichenfunktionen |
| Kernel32.dll | Basisunterstützung für 32-Bit-API des Windows-Kernels, Zuständig auch für die Speicherverwaltung |
| Lz32.dll | 32-Bit-Komprimierungsroutinen |
| Mpr.dll | Multiple Provider Router-Bibliothek |
| Netapi32.dll | 32-Bit Netzwerk-API-Bibliothek |
| Shell32.dll | 32-Bit Shell-API-Bibliothek |
| User32.dll | Bibliothek für Routinen der Benutzeroberfläche (Fenster Menüs, Maus) |
| Version.dll | Versionsbibliothek |
| Winmm.dll | Windows Multimedia-Bibliothek |
| Winspool.drv | Schnittstelle des Drucker-Spoolers, enthält die API-Aufrufe des Drucker-Spoolers |

# Anhang D Namenskonventionen und Dateikennungen

Nachfolgend sind Präfixe der in der Namengebung bei Verwendung von Steuerelementen in Visual Basic. Sie erleichtern das Lesen des Quellcodes und tragen somit zum besseren und logischen Verständnis bei. In Klammer sind Beispiele der Namensgebung genannt.

| Präfix | Steuerelement |
|---|---|
| *chk_** | Kontrollkästchen oder Check Box |
| *cmd_** | Befehlsschaltfläche |
| *frame_** | Rahmenfeld |
| *frm_** | Formular Steuerelement |
| *lbl_** | Bezeichnungsfeld |
| *lst_** | Listenfeld |
| *mnu_** | Element einer Menüleiste |
| *opt_** | Optionsfeld |
| *pic_** | Bildfeld |
| *rtf_** | Rich Text Box |
| *txt_** | Textbox |
| *Hlauf_** | Horizontale Bildlaufleiste |
| *Vlauf_** | Vertikale Bildlaufleiste |

# Literatur

[1] Peter Fröhlich
Berechnungsbibliothek Bauwesen
Excel-Arbeitsmappen für Studium und Praxis, Vieweg Verlag, 1998-X

[M1] Hans-Jürgen Holland
vieweg technic tools - Technische Mechanik I
Vieweg Verlag 1995. - Disk - Sonderformat

[2] Schmitz / Goris
DIN 1045-1 digital
Werner Verlag ,Düsseldorf , 2000

[M2] Christian Findeisen
vieweg technic tools – Massivbau
Vieweg Verlag, 1998. - Disk - Sonderformat

[3] Bernd Humbert
Excel – Arbeitsblatt zur Berechnung von Schraubenverbindungen
Fachbeitrag, bi – Journal, Werner Verlag 5/99

[M9] MS - Excel – Arbeitsmappe zur Schraubenbemessung
Diskette
Werner Verlag , Düsseldorf

[4] Konrad Kuntsche
Geotechnik Erkunden – Untersuchen – Berechnen - Messen mit Excel – Arbeitsblätter
Vieweg – Verlag, 2000

[5] Günter Schulz
Rechenbeispiel nach Schneider, Klaus-Jürgen, Mauerwerksbau
Düsseldorf: Werner Verlag 5.Auflage S.193
Fachbeitrag, bi – Journal, Werner Verlag 4/99

[6] Günter Schulz
Excel – Arbeitsblatt
Algorithmus zur Bemessung von zweiseitig gehaltenen Wänden nach
DIN 1053-1, Düsseldorf: Werner Verlag Bautabellen für Ingenieure Kapitel 7
Fachbeitrag, bi – Journal, Werner Verlag 6/98

[M7] Excel – Arbeitsblatt
Algorithmus zur Bemessung von zweiseitig gehaltenen Wänden nach DIN 1053-1
Servicediskette
bi – Journal, Werner Verlag, Düsseldorf, 6/1998

[7] Günter Schulz
Excel – Arbeitsblatt
Bemessung von Holzbalken nach DIN 1052, Düsseldorf: Werner
Verlag Bautabellen für Ingenieure Kapitel 7
Fachbeitrag, bi – Journal, Werner Verlag 1/98

[8] Günter Schulz
Kostenschätzung nach DIN 276
Fachbeitrag, bi – Journal,
Werner Verlag, 1/1999

[9] Günter Schulz
Algorithmen- und Programmstruktur der HOAI - Interpolation
Fachbeitrag, bi – Journal,
Werner Verlag, 1/1999

[10] Günter Schulz
Einfeldbalken aus Holz bemessen und nachweisen Excel – Arbeitsblatt
Fachbeitrag,
bi – Journal, Werner Verlag, 2/1999

[11] Werkle H., K.- J. Schneider
Bautabellen digital, Version 1.1a
Werner Verlag, Düsseldorf , 2000

[12] Serkan Karadag
Diplomarbeit an der FH - Konstanz
Fachrichtung Bauingenieurwesen, 03.02.2000

13] Günter Schulz
Statische Berechnungen mit Mathcad
Fachbeitrag, bi – Journal, Werner Verlag, 1/99

[14] Horst Werkle
Differentialgleichungen mit Mathcad lösen
Fachbeitrag, bi – Journal, Werner Verlag, 2/99

[15] Horst Werkle
Statische Berechnungen mit Mathcad
Fachbeitrag, bi – Journal, Werner Verlag, 3/99

[16] Kai-Uwe Jacobs, Ralf Avak
Durchstanznachweis mit Mathcad
Fachbeitrag bi – Journal, Werner Verlag 4/99

## Multimediaresourcen

[M1] Hans-Jürgen Holland
vieweg technic tools - Technische Mechanik I
Vieweg Verlag 1995. - Disk - Sonderformat

[M2] Christian Findeisen
vieweg technic tools – Massivbau
Vieweg Verlag, 1998. - Disk - Sonderformat

[M3] Volker Slowik
vieweg technic tools - Festigkeitslehre für Bauingenieure,
Vieweg Verlag 1996. - Disk – Sonderformat

[M4] Excel - Arbeitsblatt
Algorithmen- und Programmstruktur der HOAI – Interpolation
Servicediskette, bi – Journal, Werner Verlag , Düsseldorf, 1/1999

[M5] Excel - Arbeitsblatt zur Kostenschätzung nach DIN 276
Servicediskette, bi – Journal
Werner Verlag, Düsseldorf, 1/1999

[M6] Excel – Formulare zur Kostenermittlung nach DIN 276 (6.93) und DIN 276 (4.81)
Servicediskette
bi – Journal, Werner Verlag, Düsseldorf, 2/1999

[M7] Excel – Arbeitsblatt
Algorithmus zur Bemessung von zweiseitig gehaltenen Wänden nach DIN 1053-1
Servicediskette
bi – Journal, Werner Verlag, Düsseldorf, 6/1998

[M8] Excel – Arbeitsblatt
Bemessung von Holzbalken nach DIN 1052
Diskette, Bestellnr.Holz-005.XLS
Werner Verlag , Düsseldorf

[M9] MS - Excel – Arbeitsmappe zur Schraubenbemessung
Diskette
Werner Verlag , Düsseldorf

[M10] Warren C. Joung,Roark`s
Formulas for Stress and Strain
Civil Engineering Library
Math Soft ,Inc. and Mc Graw-Hill, United Kingdom, Inc. 1998

[M11] Thomas P. Magner, P.E
Building Structural Design
Civil Engineering Library
Math Soft Inc., United Kingdom, 1998

[49] Kuhr, Harald
EDV/CAD für die Bautechnik
Verlag B.G. Teubner, Stuttgart, 1991

[50] Kretzschmar, Horst
Computergestützte Bauplanung
Verlag für Bauwesen, Berlin, 1994

## URL's

[L1] Mathcad Application Files
Elastic bending in a beam, (Mathcad 4.0, 12 kb)
www.mathsoft.com//bending.mcd

[L2] Kristopher Hauck
Shear and moment diagrams for any simple/overhang
(Mathcad 7 Prof, 508 kb) -
www.mathsoft.com/ssbeam.mcd

[L3] Thomas P. Magner
Analysis of simple span beams
(Mathcad PLUS 6.0, 55 kb) -
www.mathsoft.com/smplbeam.mcd

[L4] Robert Hirschel
Theory of Plates and Shells
(Mathcad 8 Prof., 18 kb) -
www.mathsoft.com/ plate_shell.mcd

[L5] Paul H. Dunn
ConcCol - Interaction diagram for rectangular concrete columns using the CRSI formulas
( Mathcad PLUS 6.0, 330 kb)
ConcCirc - Interaction diagram for circular concrete columns using the CRSI formulas
(Mathcad PLUS 6.0, 758 kb)
www.mathsoft.com/ conccol.mcd

[L6] Richard Foss
Solid slab - tension reinforcement only
(Mathcad 8, 29 kb)
www.mathsoft.com/usd.mcd

[L7] Bill Moore
Design of circular concrete tank walls
(Mathcad 8, 96 kb)
www.mathsoft.com/ conctank.mcd

[L8] Thomas P. Magner
Strength of Rectangular Reinforced Concrete Columns
(Mathcad 8 Prof., 182 kb)
www.mathsoft.com/ rectcols.mcd

[L9] Thomas P. Magner
Reinforcing Bar Development
(Mathcad 8 Prof., 49 kb)
www.mathsoft.com/ devreinf.mcd

[L10] Bill Moore
Required thickness of a concrete slab-on-grade to resist a given applied column load
(Mathcad 6.0, 46 kb)
www.mathsoft.com/ sog_t.mcd

[L11] Sandor Dominich
Minimisation of a cost function for a welded metal joint
(Mathcad PLUS 6.0, 12 kb)
www.mathsoft.com/ weld01.mcd

[L12] Thomas P. Magner
AISC Specification Calculations
(Mathcad 7 Prof., 111 kb)
www.mathsoft.com/ AISCCalc.mcd

[L13] Wendell Day
Pole foundations without surface constraint
(Mathcad 3.1, 5 kb)
www.mathsoft.com/ pole.mcd

[L14] M. Heinloo and K. Kaul
Motion in the soil of a circular link of the free-active spinharrow
(Mathcad PLUS 6.0, 124 kb)
www.mathsoft.com/ spinharr.mcd

[L15] Thomas P. Magner
Wind loads using ASCE Standard 7-95
(Mathcad 6.0, 52 kb)
www.mathsoft.com/ windload.mcd

[L16] Thomas Schaap
Modelling hysteresis effect
(Mathcad 7 Prof., 28 kb and, MS Word document, 54 kb)
www.mathsoft.com/ elasplas.mcd , www.mathsoft.com/ elasplas.doc